FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

DEPARTAMENTO DE ENGENHARIA ELECTROTÉCNICA

Agente de Coligação com Definição Gráfica de Processos Habilitados por Executor de Regras

Execução e Composição Dinâmica de Skills num Sistema de Manufactura MultiAgente

Por:

PEDRO MIGUEL MARTINS DEUSDADO

Dissertação apresentada na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa para obtenção do grau de Mestre em Mestrado Integrado em Engenharia Electrotécnica e de Computadores

ORIENTADOR: PROF. JOSÉ ANTÓNIO BARATA DE OLIVEIRA

MONTE de CAPARICA

2011

Agente de Coligação com Definição Gráfica de Processos Habilitados por Executor de Regras

Para a minha Mãe, Pai e Irmão

Agradecimentos

Dedico este espaço a todos os que contribuíram para que esta dissertação fosse realizada. Aos meus colegas, professores, amigos e familiares, em especial:

Ao meus colegas Gonçalo Cândido e Filipe Feijão, por permitirem a utilização da infraestrutura básica que serviu de suporte ao trabalho que se apresenta nesta.

À minha amiga Paula por me motivar de forma efusiva e me ajudar com a revisão da tese. Mas também à Cláudia por me rever diversas versões intermédias da tese. Sem ambas a mesma não teria o mesmo grau de percepção.

Aos meus pais, pela compreensão, carinho, apoio, paciência, orientação pessoal e pelos valores que me transmitiram;

Ao meu irmão, pelo animo fornecido quando o desenvolvimento do projecto não corria como o desejado.

Aos meus amigos Filipe, Sofia, Mónica, Carla e Sónia pelo bem-estar criado fora do ambiente académico, sempre que era necessário passar algum tempo afastado.

Aos meus vizinhos e amigos Abílio, Filipa e Glória por me darem força para acabar a tese.

Não podia também deixar de agradecer a uma pessoa muito especial e importante, que apesar das dores de cabeça sempre me deu apoio quando precisava.

Por fim, um agradecimento especial ao meu orientador, Professor Doutor José António Barata de Oliveira por me permitir a realização desta tese.

Sem todos vós, este trabalho não seria possível.

Aqui deixo o meu agradecimento simples, mas muito sincero: *Muito Obrigado!!!*

Sumário

A globalização dos mercados e o nível de exigência imposto pelos clientes, em termos de qualidade de produtos e personalização, estão a mudar a forma como a manufactura tem de ser encarada. Como resultado surge, por parte das empresas, a necessidade de implementar processos produtivos mais ágeis, versáteis, robustos e com performance superior. Os requisitos referidos são normalmente obtidos através do uso de componentes autónomos, inteligentes e distribuídos. Uma das aproximações mais promissora ao desenvolvimento deste tipo de sistemas surge sob a forma de multi-agentes.

Uma das limitações nos sistemas multi-agentes mais utilizados hoje em dia é o facto de as entidades coordenadores não serem genéricas nem intuitivas. A definição da sequência de actividades para o grupo de agentes constituintes dessa coligação implica a necessidade de combinar conhecimentos de domínio e de programação. Sistemas de fluxogramas e regras não possuem estes requisitos pois separam a lógica do código. Porém, em ambas as tecnologias não existe suporte à noção de comunicação, interacção, autonomia e proactividade características inerentes aos agentes.

O Objectivo de criar um agente gestor onde o operador comum possa proceder à definição, modificação e execução de processos de forma simples foi o factor que levou à elaboração desta tese. Um agente de coligação com mecanismo de modelação e execução de tarefas complexas foi implementado. O agente de coligação contém um sistema de regras, destinado à dedução de tarefas complexas e um sistema de fluxogramas, requerido para a definição do processo que permite atingir a tarefa computada. Como prova de conceito foi desenvolvido um "Agente Coligador". Posteriormente o mesmo foi testado numa célula de manufactura laboratorial onde cada entidade se encontra "agentificada" e as suas capacidades se encontram disponíveis para ser descobertas e executadas.

Palavras-Chave: Sistemas Multi-Agente, Sistemas de Fluxogramas, Sistemas Baseados em Regras, Sistemas de Manufactura Reconfiguráveis, Manufactura Ágil.

Abstract

Market's globalization and boosted customer demands in terms of product quality and customization are changing the way manufacturing needs to be faced. Current manufacturing is obliged to effortlessly ensure agility, robustness and performance during (re)engineering processes. These requirements are normally achieved through the use of autonomous, intelligent and distributed components. One of the most promising approaches for the implementation of such systems is multi-agents.

An important limitation of most current multi-agent systems is absence of an intuitive and generic management entity. Defining sequences of activities for a group of agents, imposes a combination of domain knowledge and programming skills, so that the execution of new capabilities can be achieved. Workflows and Rule Based Systems do not have these requirements, since they split logic from code. However, they do not support the distribution and intelligence offered by agents.

The possibility of creating an entity where the user is able to define, modify and load the process in a user-friendly way led to this thesis. A coalition agent with improved mechanisms to model and execute complex skills was developed. The coalition Agent embeds a rule and workflow engine to support the discovery of equipment skills and definition of the required process to achieve its composition into complex skills. To prove this concept a Coalition Agent was developed, deployed and tested on a experimental manufacturing environment where each equipment module is "agentified" and its skills are available to be discovered and executed.

Key-Words: Multi Agent Systems (MAS), Workflow Systems, Rule Based Systems (RBS), Reconfigurable Manufacturing Systems (RMS), Agile manufacturing.

Índice

Índice de Figuras

Índice de Tabelas

Glossário de Abreviações

ACL – Agent Communication Language

API – Application Programming Interface

BPEL – Business Process Execution Language

BPEL – Business Process Execution Language

BPMN - Business Process Modelling Notation

CLIPS - C Language Integrated Production System

DF – Directory Facilitator

FIPA – Foundation for Intelligent Physical Agents

HMS – Holonic Manufacturing Systems

J2EE – Java 2 Enterprise Edition

JADE – Java Agent Development Environment

JESS – Java Expert System Shell

JSR-94 - Java Specification Request – Java Rule Engine API

MAS – Multi Agent System

OMG – Object Management Group

RBS – Rule Based Systems

SOA - Service Oriented Architecture

SOAP - Simple Object Access Protocol

WfMC – Workflow Management Coalition

XML – eXtended Markup Language

XPDL – XML Process Definition Language

1 Introdução

1.1 Enquadramento

Na última década, tem-se testemunhado um crescimento explosivo nas tecnologias de informação, computação e comunicação. A grande capacidade computacional, internet, conectividade e acesso universal, realidade virtual e a integração empresarial são apenas algumas faces desta revolução. Paralelamente, organizações e mercados também se alteraram dramaticamente através das organizações virtuais, integração de processos de negócio, cadeias de fornecimento centradas no cliente e comércio electrónico.

No inicio da era informática, a construção de um programa acarretava consigo a necessidade de proceder à elaboração do mesmo desde a raiz. Durante algum tempo, esta foi a única alternativa disponível e como tal era globalmente aceite. Com a necessidade de maior complexidade e menor tempo de elaboração o paradigma de programação actual deu lugar ao conceito de reprogramação. Em seu auxílio, surgem as novas linguagens e conceitos de abstracção como objectos, classes e interfaces. Graças a elas, é agora possível partilhar e reutilizar elementos de código sem ser necessário proceder a quaisquer alterações, simplifica-se assim a elaboração de novos programas. Anos mais tarde, o crescimento das empresas e a necessidade de agilizar a partilha e armazenamento de informação levou à introdução de servidores e bases de dados. Na situação actual, a flexibilidade, adaptabilidade e reutilização são pontos-chave de modo a que se consiga responder atempadamente e de forma adequada aos desafios. Como tal, a distribuição e reutilização de trabalho surgem como a solução mais adequada.

Alguns dos aspectos referidos anteriormente são passíveis de ser transportados para a área de manufactura. Aqui, a automatização começou por ocorrer de forma parcial e em algumas secções do processo. Em adição, a construção de uma nova máquina, mesmo que semelhante, iria implicar a programação quase total do sistema.

Com o aumento do grau de automatização ocorre também um acréscimo de informação a processar e armazenar pelos sistemas, como resultado as técnicas até agora utilizadas deixam de ser válidas. Numa perspectiva temporária, surgem técnicas semelhantes à área de informática marcadas pelo seu carácter centralizado.

Perante os requisitos actuais de manufactura o desafio consiste no desenvolvimento de soluções altamente reconfiguráveis e distribuídas. A importância destes conceitos é tal que são vistos como pilares da manufactura para o ano 2020 [1-3]. Uma aproximação adequada é obtida pela elaboração de componentes individuais, autónomos e inteligentes, que suportem eficazmente reconfiguração e adaptabilidade.

As implementações mais ajustadas à realidade introduzida ocorrem através dos MAS[1] e SOA[2], onde um sistema global funciona graças aos contributos de cada um dos seus membros. Na sua essência, as tecnologias anteriores são responsáveis por disponibilizar capacidades abstraindo grande parte da complexidade a elas associada. Contudo, os agentes dispõem de limitações ao nível da autonomia, nomeadamente no planeamento e escolha da decisão a tomar, na distribuição, pela existência de elementos centralizados na arquitectura, e na facilidade de implementação de determinados constituintes. Este último factor tem como causa a inexistência de um componente genérico.

O objectivo de ter uma rede constituída por entidades genéricas já se encontra instituído mas ao nível da disponibilização e publicação das operações que cada uma está habilitada a efectuar. A implementação genérica dos diversos agentes constituintes de um sistema de manufactura também já foi alvo de estudo num trabalho que serviu de base ao sistema apresentado [4]. Não obstante a elaboração de entidades coordenadoras permanecia sem uma implementação de versatilidade equivalente. Estes agentes de *software*, habilitados a gerir agrupamentos de agentes mais simples são os responsáveis pelo surgimento de novas capacidades de execução (skills) mais complexas.

A existência de uma versão genérica destas entidades, denominadas de agentes de coligação, é de extrema importância na obtenção dos requisitos de manufactura actuais. Através desta nova implementação será possível obter novas capacidades aquando da adição de novos elementos.

Através da Figura 1.1 é possível depreender algumas vantagens desta nova aproximação. Ao combinar três tecnologias aparentemente distintas - Agentes, Regras e Fluxogramas - consegue-se por um lado proceder à definição de forma intuitiva e sem recurso à programação da sequência de operações necessárias à realização de uma determinada acção. Por outro lado permite-se a enunciação de diversas condições destinadas a determinar quais dessas acções se encontram habilitadas a ser executadas.

Como a facilidade de construção, modificação e reutilização sempre foi um dos requisitos pretendidos para este sistema estamos perante uma aproximação que se nos afigura adequada.

Este projecto propõe um agente coordenador genérico com a capacidade de executar um conjunto de actividades definidas e estruturadas num modelo gráfico. Para o conseguir integrou-se um sistema de regras e um executor de fluxogramas numa entidade agente básica.

[1] Sistemas MultiAgente
[2] Arquitecturas Orientadas a Serviços

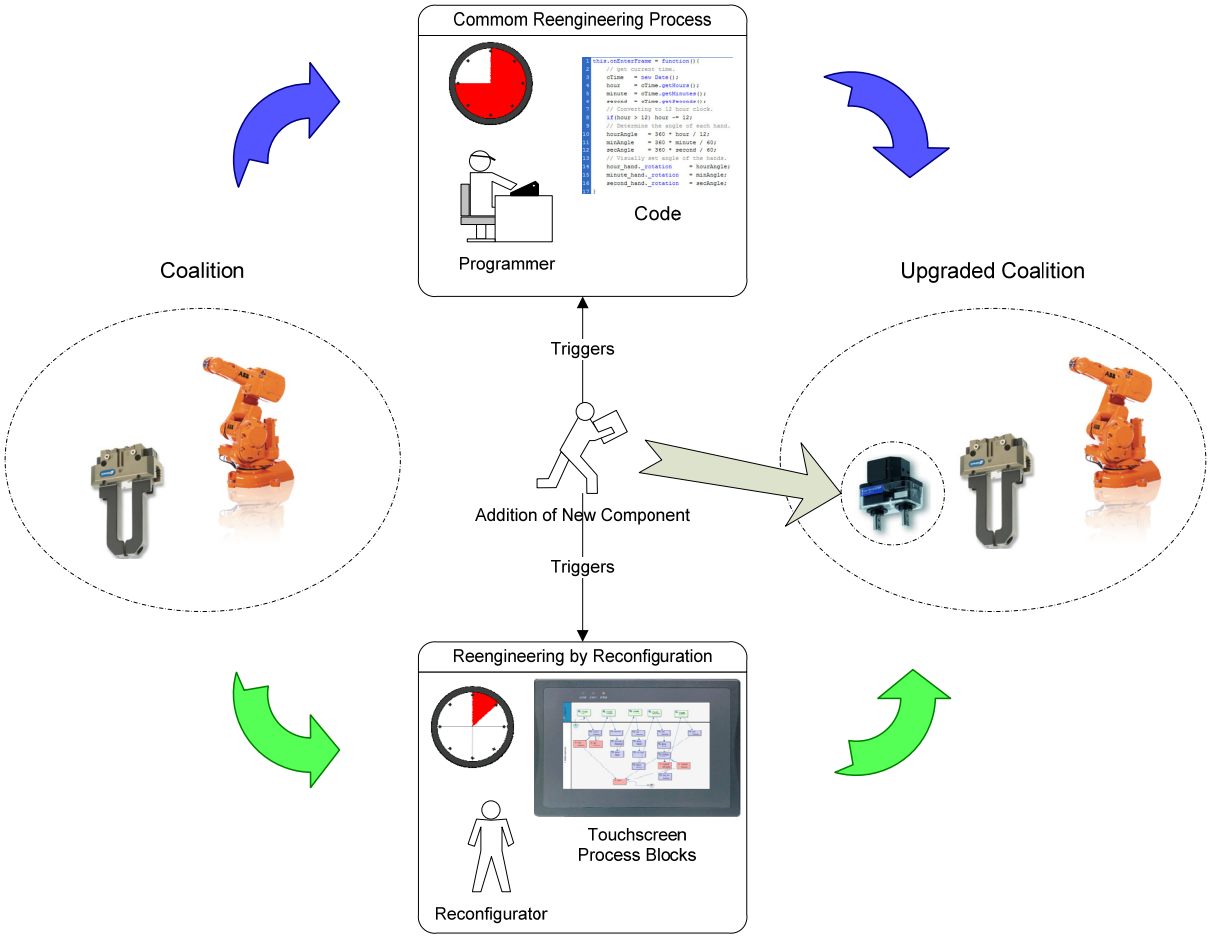

Figura 1.1- Processo de Reengenharia Antigo vs. Processo de Reengenharia proposto. Em ambas as situações a adição de um novo componente é o factor responsavel pelo desencadear de operações. Contudo no segundo método caso o sistema não disponha já da informação necessária o utilizador pode facilmente proceder à definição manual.

A Figura 1.2 mostra, de forma sucinta, como os diversos elementos foram combinados de modo a dar origem à arquitectura que se considera mais adequada.

O agente, entidade representativa de todos os elementos físicos do sistema (*Gripper*, *ToolWareHouse*, *Robot*, *Conveyor*), tem na coligação o papel de infra-estrutura de suporte às restantes tecnologias utilizadas. O encapsulamento por ele fornecido não só disponibiliza todos os métodos de comunicação úteis, por exemplo, na recepção de pedidos e entrega de actividades a realizar pelas entidades remotas, como também o torna indiferenciável relativamente a qualquer outro agente.

O executor de regras incorporado recebe e processa ficheiros de regras. As condições neles descritas, quando combinadas com as capacidades das diversas entidades constituintes da coligação habilitam a instanciação e execução de processos por parte do executor de fluxogramas.

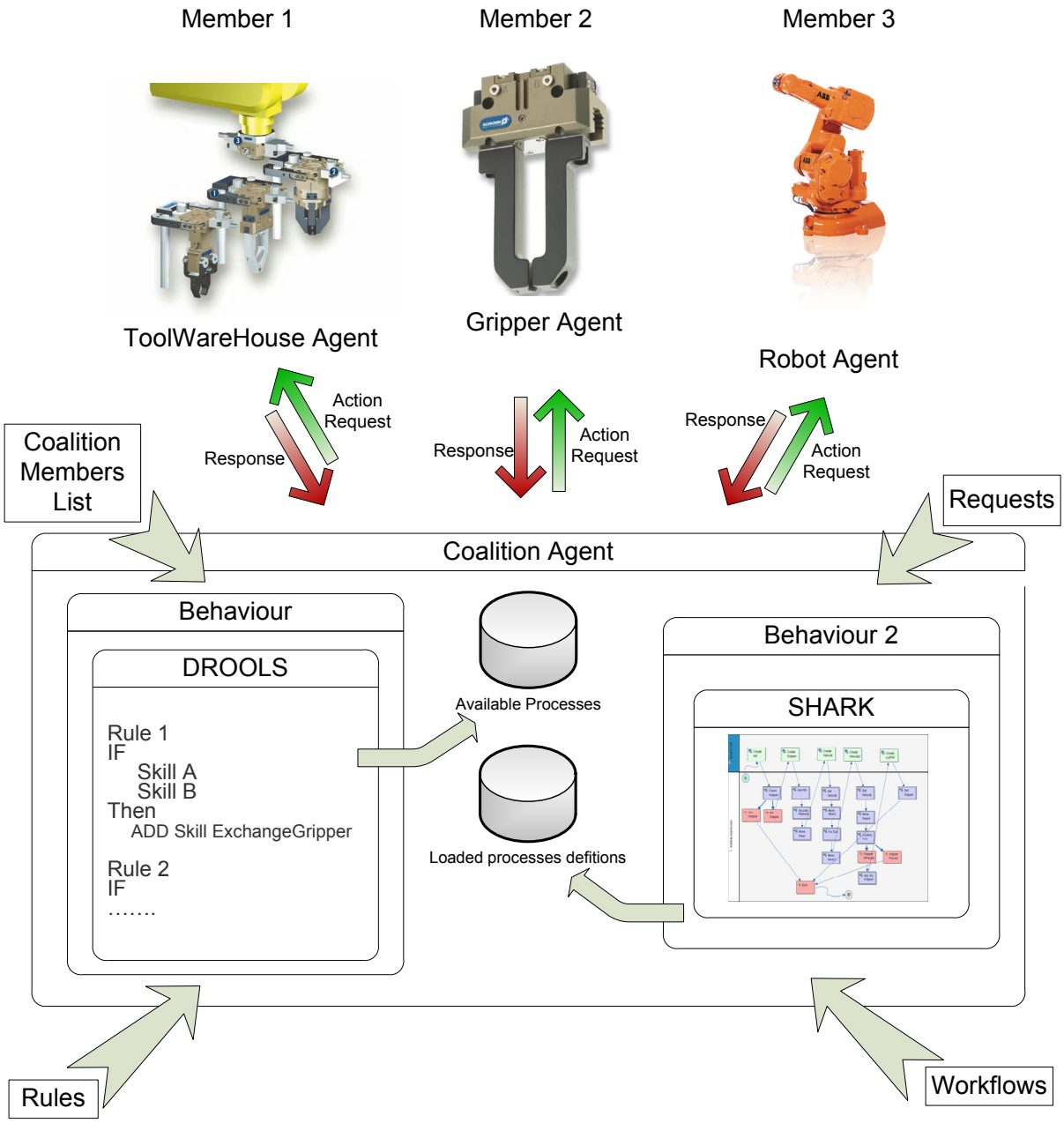

Figura 1.2 - Arquitectura global do agente coligação proposto onde se apresentam também alguns dos membros com os quais irá comunica. O sistema de regras determina quais os processos são passíveis de ser executados e guarda-os na lista "Available Processes". O executor de fluxogramas carrega o ficheiro indicado na sua interface e guarda os processos nele definidos na lista "Loaded processes definitions". Mais tarde, aquando de um pedido, ambas serão consultadas.

Por último, quando habilitado, o executor de fluxogramas instancia o processo requerido e determina qual a actividade a executar. Os parâmetros necessários à sua concretização são recolhidos pelo agente coligação e entregues à entidade remota adequada. Os resultados recebidos são devolvidos ao executor e utilizados na determinação da próxima acção a executar.

De modo a garantir os requisitos indicados antes, a escolha de aplicações adequadas teve de respeitar critérios como versatilidade, simplicidade e intuitividade. O construtor de fluxogramas dispõe de uma interface gráfica muito acessível que permite a construção de processos por arraste de

blocos. A definição das propriedades necessárias é também obtida naturalmente ao utilizador de computador comum através dos menus de contexto. O editor de regras também se coíbe com os critérios anteriores ao dispor de ajuda contextual e possibilitar a utilização de linguagem de domínio, facilidades disponíveis quando se recorre à alternativa textual. Existe também a opção de definição de regras através de ficheiro Excel o que eleva a versatilidade a outro nível.

No final, o agente construído irá permitir a um utilizador comum, sem conhecimentos extensos de programação, proceder à definição/configuração de operações através de blocos e de regras responsáveis por verificar que a sua instanciação é possível.

Para provar o conceito, será desenvolvido um agente coordenador onde foi agregado o sistema de regras e um executor de fluxogramas.

1.2 Sumário da Dissertação

Esta tese encontra-se organizada em seis capítulos: Introdução, Estado da Arte, Tecnologias de Suporte, Arquitectura, Implementação e Epílogo.

Capítulo 1 (Introdução) – Introduz o problema no domínio adequado e descreve de forma sucinta o componente implementado e as suas funcionalidades principais.

Capítulo 2 (Estado da Arte) – Introduz a evolução do conceito de manufactura. Apresenta também sobre este, paradigmas do passado e presente com incidência nas vantagens e desvantagens de cada um. Contextualiza os agentes nos paradigmas de manufactura actuais.

Capítulo 3 (Tecnologias de Suporte) - Expõe as diversas características, constituintes e vantagens dos diversos elementos constituintes da aplicação final. Os conceitos aqui introduzidos apesar de não se encontrarem direccionados para a implementação específica pretendem ser uma fonte de informação global e valiosa para este tipo de sistemas. Descreve também os diversos *standards* seguidos no trabalho, alguns dos quais são utilizados noutras áreas.

Capítulo 4 (Arquitectura) – Neste capítulo é introduzida a arquitectura actual, junto com a arquitectura do sistema envolvente. São também referidas algumas das transformações que levaram ao modelo corrente.

Capítulo 5 (Implementação) – São apresentadas algumas experimentações efectuadas e as alternativas de sistemas analisadas. No final, é disponibilizada toda a informação relevante sobre a implementação realizada.

Capítulo 6 (Epílogo) – São enunciados os resultados obtidos pela solução apresentada e sugeridas direcções de trabalho futuro.

2 Estado da Arte

2.1 Manufactura

Desde cedo que a manufactura de produtos e bens é vista como geradora de riqueza[5]. As suas origens remontam à "produção artesanal", realizada em pequenas lojas, onde a criação é feita por unidades e as quantidades são pequenas. Os artesãos, indivíduos altamente habilitados e auxiliados por ferramentas gerais, socorrem-se da sua experiência e conhecimento na reprodução exacta das especificações do cliente.

A revolução industrial, impulsionada pela invenção da máquina a vapor e de combustão, introduz a maquinaria. Inicialmente destinada a ajudar os artesãos nas suas tarefas, evolui e dá origem às linhas de produção e em consequência introduz um novo paradigma - produção em massa [6, 7] – caracterizada de entre as diversas particularidades pela sua rigidez.

O conceito de competitividade nasce como resultado das alterações sociais e das crescentes flutuações do mercado motivadas quer por factores externos quer pelo próprio tipo de produção. Juntamente com o aumento do poder de compra do consumidor, que exigia agora mais variedade e qualidade, e a influência da electrónica e dos computadores são alguns dos factores responsáveis pela substituição da estabilidade e previsibilidade outrora oferecida pela diversidade e turbulência que caracteriza este conceito [8]. Segundo Hitomi, as razões anteriores permanecem válidas nos dias de hoje [9]. Devido à competitividade assiste-se a uma mudança do processo produtivo onde a anterior mecanização é substituída por automação. A evolução descrita provoca a redução de postos de trabalho, quer por razões de ordem tecnológica, pois as máquinas laboraram sem parar, ou económica, visto que o custo de produção por unidade é mais baixo. O resultado? Desagrado geral por este tipo de produção. A conjunção de todos estes elementos revelou a necessidade de um novo paradigma – *lean manufacturing* [10].

O seu objectivo de eliminar o desperdício, definido como *Muda*, e aumentar a velocidade de resposta é conseguido com a introdução do conceito de grupos de trabalhadores, continuamente empenhados em manter e melhorar o processo de forma a reduzir defeitos - TQM[3], e a redução de inventários, através do JIT[4] e Kaban[5], técnicas destinadas à sincronização entre fornecedores e distribuidores. No entanto, a adesão não foi a esperada e ocorreu maioritariamente na área automóvel e de electrónica onde o volume é grande e a variedade é pequena [11, 12].

[3] Total Quality Management
[4] Just In Time
[5] Sistema de planeamento associado ao JIT

Os paradigmas/técnicas anteriores são marcados por disponibilizarem flexibilidade apenas ao nível do controlo e supervisão de entidades máquina. Uma resposta adequada às flutuações e competitividade crescente irá requerer a visualização de cada máquina como um agrupamento de entidades modulares, mais pequenas, que colaboram entre si [13].

Na era actual de mercados globais, imprevisíveis, intensamente concorrenciais, sujeitos a grandes flutuações e perturbações onde os produtos são mais complexos, altamente personalizados, e com ciclos de vida mais curtos, as empresas só sobrevivem e estão acima da concorrência se fornecerem uma reposta rápida aos requisitos do cliente - produto certo na hora certa. Como tal, é necessário que se adaptem e modifiquem. Tal só é possível graças às evoluções a ocorrer em níveis como o conceptual, de *software* ou *hardware*.

Nos últimos tempos, constatou-se também uma grande evolução em termos de paradigmas, motivada pelos requisitos de manufactura com que agora nos deparamos. Agilidade, reconfiguração e flexibilidade são termos frequentemente utilizados e servem como pano de fundo a novos modelos da indústria como Sistemas de Manufactura Biónicos (BMS)[14] , Sistemas de Manufactura Holónico (HMS)[15-18] , Sistemas de Manufactura Reconfiguráveis (RMS)[19, 20] , Sistemas de Assemblagem Evolutivos (EAS)[21-23], Sistemas de Produção Evolutivos (EPS)[24, 25]. Por apresentarem pontos semelhantes, a combinação de funcionalidades é possível e recomendável. Consegue-se assim a maximização de vantagens.

Contrariamente ao que se poderia pensar e apesar do enorme potencial, a sua adopção tem decorrido de forma muito lenta, acontecimento motivado por diversos motivos.

Uma das razões relaciona-se com o facto de a indústria apenas adoptar teorias que compreende e se encontrem bem definidas. Nestas porém, não existe uma definição clara das funcionalidades que irão ser disponibilizadas, nomeadamente autonomia, auto-organização e emergência, nem um entendimento de como os sistemas adaptativos funcionam. A aceitação destes novos modelos irá implicar um método inovador de abordar os problemas e uma nova forma de pensar, o que não será uma tarefa fácil.

A ausência de um componente central, marcante em todas as arquitecturas anteriores, provoca desconforto e traz ao de cima o medo por algo incontrolável. É imperativo convencer possíveis investidores que estas teorias apesar de elevarem o grau de autonomia, o controlo final decai sempre sobre o utilizador[26, 27].

Outro dos motivos utilizados de forma mais frequente prende-se com o conservadorismo associado à indústria. Como Hall[28] refere, clientes e indústria apenas querem utilizar tecnologia comprovada. No entanto, não pretendem ser os primeiros a experimentá-la nos seus processos [29,

30]. É então essencial a demonstração de que se está perante tecnologia consolidada, passível de ser aplicada com vantagens [31]. Este factor recebe ainda mais relevo quando se prevê que o dinamismo futuro requisitado pelo mercado seja o factor que obrigue à mudança.

O esforço e investimento adicional associado ao requisito de modularidade necessário à adaptabilidade sem que, ao contrário dos métodos tradicionais flexíveis, exista garantias quanto à utilização dos benefícios disponibilizados. É pois importante levar a indústria a pensar para além do óbvio e imediato. Se necessário demonstrar, através da natureza, os ganhos susceptíveis de serem obtidos no mercado turbulento em que se inserem [32].

A integração dos novos paradigmas com sistemas e aplicações já existentes é também um elemento de grande influência. Os sistemas actuais destinados a permitir tais operações, de forma sistemática, não se encontram ainda implementados uma vez que as tecnologias necessárias se encontram na fase de desenvolvimento [33]. Em situações deste tipo a solução passa por utilização de interfaces de mediação. MAS e SOA afiguram-se como respostas promissoras devido à sua independência em termos de plataforma.

De entre os múltiplos motivos ainda por referir o último a merecer especial destaque está relacionado com dinamismo, pensamento e aprendizagem distribuídos associado a cada um dos modelos. A alguns deles vem também associada a palavra emergência que provoca receio pois inerente a ela surge a imprevisibilidade temida pela indústria. É necessário mostrar de forma simples e com implementações reais os conceitos enunciados antes, de modo a serem aceites como triviais [26, 27].

O número de sistemas a ser desenvolvido em torno dos paradigmas referidos, vistos como solução às necessidades de manufactura actuais, é extenso. A mencionar o ADACOR – aproximação holónica para controlo de manufactura [11]; COBASA – aproximação baseada em agentes destinada a melhorar a agilidade e reconfiguração da unidade fabril [12], iShopFloor – aproximação baseada em agentes destinada a criar ambiente *Plug and Play* em grandes redes [34]; ARTIS – arquitectura baseada em agentes destinada a aplicações de tempo real [35], entre muitos outros.

A investigação relacionada com a obtenção de reconfiguração, agilidade e evolutividade na indústria tem progredido de forma rápida nos últimos anos. Para tal, projectos como o EUPASS[6] - Sistemas de ASsemblagem Evolutivos de alta precisão ou o PABADIS PROMISE[7] – sistemas de manufactura orientados ao produto para empresas reconfiguráveis [36], têm dado um grande contributo. Porém, é necessário mais trabalho até que essa meta seja atingida.

[6] Evolvable Ultra Precision Assembly Systems
[7] PABADIS based Product Oriented Manufacturing Systems for Re-Configurable Enterprises

A modularidade é elemento fulcral em grande parte das alternativas apontadas e possibilita o aumentar da variedade de produtos. Junto com as futuras arquitecturas abertas e a superior coordenação entre os diversos elementos da cadeia ira permitir melhor actualização e eficiência de custo. Está também presente no requisito de *zero down time* e no de fiabilidade pois junto com o diagnostico é necessário na rápida detecção e resolução do problema. Posteriormente, a maior velocidade de resposta implica a introdução de ferramentas de análise e gestão de risco nos sistemas que procedem à estruturação do processo. Neste grupo de desafios onde se inclui o suporte à heterogeneidade em todos os níveis, e a partilha e tradução de conhecimento, interessa ainda referir a importância de criar um mecanismo de detecção da necessidade de evolução ou reconfiguração [13, 37-40].

Nesta altura, depreende-se com facilidade que a imprescindível reconfiguração é tanto mais eficaz e eficiente quanto maior for a colaboração entre os diversos níveis de uma empresa. Essa propriedade leva, por exemplo, Stechi e Lagos[5] a optarem por apresentar os desafios a diferentes níveis de abstracção. Segundo eles, o *Hardware* necessita de robustez de modo a funcionar de forma fiável e segura em diversas condições e configurações. Os sistemas de controlo requerem flexibilidade e possibilidade de programação total. O *software* carece de personalização, abertura, modularidade e dinamismo com intuito de possibilitar *Plug and Play*. O Sistema reclama por metodologias de desenho, destinadas a desenvolver os sistemas de acordo com os requisitos do cliente e critérios de avaliação das diferentes configurações ao nível da fiabilidade, qualidade e custo.

O expoente mais elevado da reconfiguração será atingido quando as empresas deixarem de ser consideradas como entidades individuais e a colaboração se estender além dos diversos departamentos que as constituem. Esta medida irá implicar, entre outros, a sincronização de actividades entre as diferentes entidades.

Independentemente dos factores ou categorizações apresentados há uma actividade que não pode ser descurada – Demonstração - pois permite a prova de conceitos teóricos e serve de testemunho à ocorrência de evolução.

2.2 Paradigmas da Manufactura

A participação crescente dos consumidores nos processos produtivos por exigência de características particulares e menor tempo de entrega tornam as aproximações tradicionais, centralizadas, ineficientes devido à falta de flexibilidade, robustez e reconfigurabilidade. A biologia e natureza surgem como fonte de inspiração adequada ao desenvolvimento deste novo tipo de sistema onde a complexidade se encontra distribuída, cada uma das unidades possui autonomia e inteligência, ainda que limitada, onde as decisões são tomadas em grupo e os objectivos atingidos por colaboração [41].

Paradigmas recentes, como Sistema Manufactura Fractais (FMS ou FF), Sistemas de Manufactura Biónicos (BMS), Sistemas de Manufactura Holónicos (HMS), Sistemas de Manufactura Reconfiguráveis (RMS), Sistemas de Assemblagem Evolutivos (EAS), Sistemas de Produção Evolutivos (EPS), munidos das capacidades necessárias, vêm em nosso auxílio na resolução dos problemas indicados.

O termo **Fractal** provém da teoria do caos e da geometria fractal, utilizada na descrição e análise de objectos em espaços multi-dimensionais. A sua principal característica é a auto-similaridade, o que envolve conceitos como recursividade e *padrões dentro de padrõ*es[42]. Em termos práticos traduz-se no facto que cada entidade possui um conjunto similar de componentes e partilha um grupo de objectivos – exteriores iguais, interiores diferentes (Figura 2.1). Essa propriedade é aumentada com a auto-organização. Definida como a capacidade do sistema se reorganizar, evoluir e tomar decisões sem intervenção externa e segundo critérios nela embutidos, aparece muitas vezes associada ao dinamismo. A Auto-optimização também está presente e procura continuamente obter a melhor *performance*.

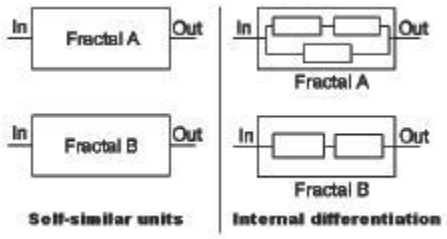

Figura 2.1 – Conceito base de um sistema Fractal [43]

Conceito de fábrica fractal pretende obter características semelhantes na indústria. Nela as empresas são constituídas por pequenas entidades denominadas de objectos fractais que cooperam entre si. Como resultado das suas características advém um alto dinamismo individual e capacidade de reacção e adaptação rápida a alterações do seu ambiente – Vitalidade [44].

O facto dos fractais constituintes não terem uma organização pré definida e de agirem como entidades individuais em busca dos seus objectivos são alguns dos motivos que levam às propriedades anteriores. No entanto, o funcionamento como um todo é o pretendido. Tal é conseguido através de comunicação e coordenação suportado por um mecanismo de herança que assegura a consistência do objectivo [41].

Segundo Tharumarajah [43] é possível encontrar muitas características isoladas da fábrica fractal no dia-a-dia. Existem também algumas aplicações reportadas, como é o caso de Mettler-Toledo que propõe fractais auto-gestores no desenvolvimento de novos produtos e tecnologias e na gestão de

produção. Segundo eles, a disposição variável por eles conseguida, junto com a utilização flexível de pessoal, permite aumentar a adaptabilidade a condições de mercado variáveis.

A palavra **Biónica** provém da combinação da palavra grega "βίον"[8] – unidade viva, com o sufixo "-ic"[9] – no sentido de, o que resulta no "como unidade viva"[10]; alguns dicionários apresentam-na como resultado da junção entre **bio**logia e electró**nica - Biónica.** É definida como o estudo e análise dos princípios básicos encontrados na natureza e a sua incorporação no desenho de *hardware* e sistemas de engenharia [45].

Este tipo de sistemas efectua o paralelismo entre os sistemas biológicos e de manufactura ao nível da estrutura e organização comportamental dos organismos vivos. Deste modo, consegue-se a exibição de vantagens como a "estabilidade", autonomia, comportamento espontâneo e harmonia social tão marcantes e desejadas na manufactura [42, 46]. Segundo Saadat[8] a sua principal característica é auto-organização, auto-aprendizagem.

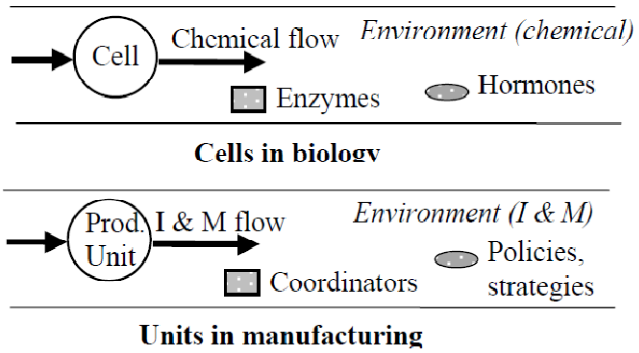

Figura 2.2 – Semelhanças entre as estruturas biológicas e de manufactura [43].

A Figura 2.2 permite traçar, quer a nível de funcionalidades como de entidades, diversas equivalências entre os dois meios. A célula, entidade básica do ambiente biológico, é equiparada à unidade de produção. O ambiente químico, em que se encontra emersa e interage, é similar ao ambiente de informação e material na indústria. O seu controlo, efectuado por enzimas, garante harmonia face a entidades autónomas e encontra equivalência nos coordenadores. A capacidade evolutiva a elas associada (divisão de células, replicação DNA) é obtida pela divisão/reagrupamento dos sistemas de manufactura em unidades funcionais mais pequenas, segundo a informação proveniente da propagação dos níveis superiores. Perante este elemento constata-se que a hierarquização, com suporte dos níveis adjacentes, e o subsequente transporte de informação entre eles também ocorre de forma similar nos dois sistemas.

[8] Unit of Life
[9] Like
[10] Like Life

As analogias apresentadas levam à criação de um elemento modelador que lhes sirva de suporte – *modelon* – destinado a concretizar todos os recursos e produtos. Cada um deles possui propriedades estáticas (memoria) e comportamentos (operadores) que quando combinados com os de outro dão origem a novos *modelons* segundo uma estrutura hierárquica [14, 39]. Okino [47, 48] utiliza o conceito introduzido como base da sua visão de fabrica Biónica, onde mostra muitos dos paralelismos apresentados antes**.** A informação detida por cada um é comparada à informação genética. Deste modo são os próprios modelons que contêm a forma de realizar operações ou de como efectuar o processamento. Associado a esse conhecimento surge a noção de morfologia, na qual o modelon superior detentor da especificação procede à selecção e entrega de propriedades e comportamentos aos modelons sobre a sua alçada a fim de proceder à sua realização. A autonomia do processo anterior pode dar origem a conflitos, logo recorre-se a coordenadores no controlo e regulação do sistema e na imposição de regras organizacionais de modo a ser possível executar as tarefas pretendidas.

O vocábulo **Holon** foi introduzido pelo filósofo Koestler [49] durante os seus estudos e define a natureza híbrida dos organismos vivos e das organizações sociais. O termo resulta da combinação da palavra Grega "Holos" - tudo com o sufixo "on" que tal como o protão - *prot**on*** ou neutrão - *neutr**on***, na ciência quântica, sugere parte ou partícula. Segundo o consórcio dos Sistemas de Manufactura Holónicos (HMS), inserido no programa de Sistemas de Manufactura Inteligente [50], é definido como *"bloco de construção dos sistemas de manufactura, autónomo e cooperativo, destinado a transportar, armazenar e/ou validar informação ou objectos físicos."*

Através dela retiram-se como principais características dos holons a autonomia e a cooperação atribuídas por Tharumarajah [43] à sua natureza híbrida (todo/parte). Tal sucede-se porque a primeira possibilita a decisão de como executar o trabalho (auto-regulação) em oposição, a cooperação mostra a disponibilidade em ser inserido num grupo o que implica negociação. Outra particularidade proveniente desta dualidade é conhecida como efeito de *Janus*. Baseia-se no princípio presente na definição onde "… *um holon pode ser parte de outro holon*" (Figura 2.3) e tem como função a redução da complexidade do problema. Em adição um holon inclui na sua constituição capacidades operacionais, conhecimento, utilizado na tomada de decisões, objectivos individuais e pode representar qualquer entidade lógica ou física, onde se incluem os humanos.

Outro termo a ter em conta quando falamos de holons é *Holarchy,* definida como "*um sistema de holons passíveis de cooperarem a fim de atingir um determinado objectivo e onde as regras básicas desse processo são definidas por ela.*"

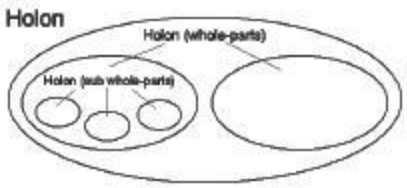

Figura 2.3 - Arquitectura de referência de um sistema holónico [43]

O consórcio adapta os conceitos desenvolvidos por Koestler no desenvolvimento dos HMS. Segundo esta entidade um Sistema de Manufactura Holónico é "*uma holarchy que integra a gama completa de acções desde concepção, produção a marketing de forma a concretizar uma empresa de manufactura ágil*". A definição anterior reforça a ideia de Saadat [8] sobre a importância dada por estes sistemas à concepção da arquitectura de controlo, a qual é do tipo distribuído e onde a unidade básica – holon - materializa cada unidade de produção. Estes por sua vez estão habilitados a cooperar entre si e a formar coligações de forma a atingir os objectivos do sistema.

Os sistemas de manufactura Holónicos tentam assim juntar o melhor dos dois mundos. Por um lado a rapidez de resposta e robustez das organizações descentralizadas, por outro a estabilidade e eficiência das arquitecturas baseadas no conceito de hierarquias [51].

Todos os paradigmas introduzidos procuram obter distribuição, autonomia e adaptabilidade. Os métodos utilizados na aquisição de cada um dos objectivos são distintos entre as diversas alternativas. Por vezes, surgem implementações que combinam características das diferentes concepções. O BMS e em especial o HMS são os conceitos mais aptos às metodologias de programação actuais, como sejam os objectos. Os seus constituintes *modelons* e *holons* exibem algumas similaridades. No entanto, os *holons* e a sua organização em *holarchy* exibem a noção de todo e parte contrastante com a ideia de níveis do BMS [38, 42].

Segundo Bussmann [52, 53] um sistema MAS é uma concretização de um sistema Holónico, onde os *holons* podem ser vistos como agentes e as *holarchy* como coligações destinadas a atingir os objectivos do sistema.

2.3 Agentes

Os conceitos de Agente e MAS emergiram de um conjunto de disciplinas que incluem a inteligência artificial e a concepção e análise de sistemas usando a metodologia orientada a objectos e interfaces humanas [54]. Representam uma área relativamente recente na ciência computacional que teve as suas origens, ainda na forma individual, com o modelo de computação ACTOR proposto por Hewitt, no seu trabalho sobre o sistema PLANNER [55]. Contudo, só nos anos 90 a investigação sobre os MAS ganhou relevo.

A definição de agente não é única nem consensual pois cada uma dá ênfase aos aspectos sobre o qual a investigação se debruçou [56-59]. Uma alternativa possível é dada por Jennings e

Wooldridge [60, 61] "*um agente é considerado uma entidade de software situada num ambiente de produção, com inteligência capaz de acções de controlo autónomas nesse ambiente e de construir relações de cooperação ao participar em associações (acordo consensual) com outras entidades de forma a atingir os seus objectivos de concepção. Um agente deverá ser capaz de agir sem intervenção directa dos humanos ou outros agentes, e deverá ter controlo sobre as suas acções e estado interno.*"

Apesar da ausência de consenso relativa à definição existe um conjunto de características cuja presença se considera importante de modo a que o agente possa ser considerado inteligente [12, 57]:

Continuidade temporal – Um agente deve ter existência contínua e longa, onde desenvolve as suas funções;

Autonomia – Um agente é autónomo quando consegue tomar decisões baseadas na sua experiência e sem intervenção externa. Uma particularidade da autonomia é permitir a sobrevivência da entidade mesmo face a mudanças drásticas do ambiente;

Sociabilidade – Um agente deve ser capaz de comunicar com outros agentes ou entidades, característica que irá implicar posterior negociação ou cooperação. Dessa interacção poderão surgir comportamentos emergentes;

Racionalidade – O agente deve ser capaz de "raciocinar" sobre os dados recebidos de modo a obter a resposta mais adequada;

Reactividade – Capacidade de o agente responder, em tempo útil, às alterações que ocorrem no ambiente, por alteração do comportamento;

Proactividade - Um agente é proactivo quando consegue exercer algum tipo de influência sobre os comportamentos consoante a sua agenda e objectivos;

Adaptabilidade – Capacidade do agente aprender e de com esse saber realizar adaptação ao ambiente onde se encontra inserido e às alterações que nele ocorram.

Mobilidade – Capacidade do agente se mover na rede de agentes sem perda do seu conhecimento e estado interno.

Existem, todavia, divergências quanto à importância de cada uma destas propriedades. O desacordo de opiniões e a multiplicidade de alternativas passíveis de serem criadas leva à necessidade de criação de classificações. Algumas das muitas possibilidades estão disponíveis em [11, 12, 57, 62, 63].

Numa tentativa de clarificar esta situação Wooldridge e Jennings dividem os agentes em deliberativos, reactivos, e híbridos. Nos deliberativos os agentes são vistos como entidades capazes de raciocinar e tomar decisões segundo o seu modelo do mundo externo. Desse "processamento" resulta um plano de acções a ser executado com vista a atingir os objectivos nele incutidos. Os reactivos partem do princípio que na obtenção de um comportamento inteligente não é necessária uma representação do meio ou um processo de raciocínio, as suas acções são o resultado directo de estímulos do ambiente. Ao contrário do anterior, aqui, procura-se a robustez em alternativa à

eficiência. Na grande maioria das situações, os extremos nunca são a solução adequada. A arquitectura híbrida resolve esse problema ao combinar o melhor da deliberação e reactividade. Consegue-se assim rápida resposta e geração de sequências de acções óptimas.

Um tipo de agente deliberativo bastante conhecido baseia-se no modelo Crenças, Desejos e Intenções - BDI[11]. A sua constituição inclui as Crenças representativas do conhecimento sobre o ambiente externo, Desejos indicativos dos objectivos a ser atingidos a longo prazo, e Intenções detentoras do plano de acções a realizar no imediato de forma a conseguir atingir os objectivos intermédios [64]. O seu funcionamento pode ser resumido a um conjunto de operações, iniciado pela exposição ao meio envolvente. Dessa exposição é retirada informação com propósito de criar uma representação do mesmo (crenças), e por combinação com as intenções actualizam-se os desejos. Da combinação dos três elementos, sucede-se a actualização das intenções, e a sua posterior utilização na selecção do melhor plano a executar. Com este modelo oferece-se ao agente a possibilidade de adicionar ou retirar objectivos determinantes na definição das próximas acções a executar.

No entanto, o conceito de Agente introduzido tem pouco interesse a título individual. Tal sucede-se porque na grande maioria das vezes não consegue resolver o problema ou efectuar o trabalho requisitado a termo individual. Nessas situações, os MultiAgentes surgem como solução. Com intuito de simplificar o processo de construção destas aplicações e reduzir o esforço de programação associado existe um conjunto de plataformas que implementam as funcionalidades associadas a esta tecnologia, como seja o transporte de mensagens ou o directório de facilidades (DF). JADE [65], RETSINA [66], FIPA-OS [67] Agent Builder [68] são alguns desses sistemas. Esta multiplicidade de alternativas faz crescer o receio de problemas ao nível da interoperabilidade entre sistemas. Na resolução dessa questão *standards* como o FIPA desempenham um papel essencial [12]. O verdadeiro desafio destas soluções estende-se para além dos sistemas actuais com a integração de tecnologias em diferentes fases do seu ciclo de vida. Actualmente, os agentes solucionam essa situação com a utilização de um agente que encapsula o recurso e assim possibilita a comunicação e cooperação com outras entidades [64]. A importância deste assunto faz com que continue a ser visto como um desafio futuro.

2.4 MultiAgentes

O desenvolvimento do conceito de MAS começou por se encontrar distanciado do seu congénere, período durante o qual se focou nos sistemas periciais e na interacção agente – humano. Só mais tarde, durante os anos 90 com o aparecimento da internet, o paradigma foi amplamente estudado e as interacções agente - agente consideradas [69].

[11] Beliefs, Desires, Intentions

A natureza descentralizada, autónoma e colaborativa dos sistemas MAS permite expor competências como a flexibilidade, robustez, reconfiguração/reutilização, *Plug and Play*, e aprendizagem. A combinação dos aspectos anteriores torna estes sistemas candidatos adequados à utilização em sistemas de manufactura futuros [39, 61, 70] [69].

Definido segundo Lee [71], como uma "*rede sem ligações rígidas de solucionadores de problemas, a trabalhar em conjunto na resolução de problemas para além das suas capacidades individuais*". Cada entidade constituinte é concretizada por um Agente (conceito já introduzido), envolvido num determinado ambiente, sobre o qual apenas recai de uma visão e controlo parcial. A representação efectuada por cada um deles pode não ser idêntica e depende de vários factores. São construídos com determinado propósito ou propósitos e por isso tem objectivos a atingir, ainda que limitados em termos de meios [57].

Pelos diversos aspectos abordados até ao momento deduz-se com facilidade a importância da interacção nos MAS. Conjugada com a inteligência e capacidade de processamento de cada entidade permite a resolução de problemas além das competências individuais. É, assim, exibido o comportamento denominado emergente, mais complexo, não previsto na fase de construção e cujo resultado é maior que a soma das suas partes [12]. Outras referências para este assunto [24, 72-74].

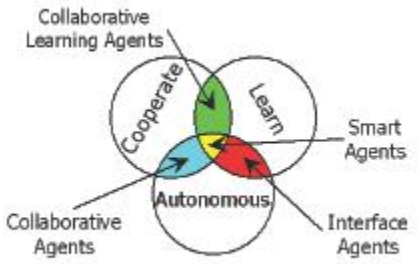

Figura 2.4 - Correspondência entre tipos de agentes e suas características [75]

Na realidade se cada um dos membros executar as suas tarefas, a soma das suas actividades poderá resultar no caos em alternativa ao "comportamento superior" indicado antes. A cooperação, colaboração, coordenação, negociação têm por isso papéis fundamentais pois possibilitam entre outras, a concretização do objectivo comum, a concertação de actividades das diversas entidades e a resolução de conflitos comuns (Figura 2.4) [74, 76].

Num sistema MAS, a comunicação/interacção que torna possível os mecanismos anteriores pode ser realizada de forma indirecta através do ambiente ou por troca directa de informação entre agentes. O conceito pressupõe o entendimento entre os seus intervenientes. No segundo caso, tal é possível com a utilização de linguagem, conceitos e protocolos de interacção comuns, dependentes do domínio onde se encontram inseridos.

As linguagens de comunicação de agentes – ACL, contribuem na resolução do problema apresentado ao tornarem transparente a troca de informação entre agentes. O seu papel é essencial no objectivo de *standardização* das mensagens trocadas durante o processo comunicativo. A primeira a ser criada foi a Linguagem de Questionamento e Manipulação de Conhecimento - KQML[12] à qual se seguiu a Fundação para Agentes Físicos Inteligentes - FIPA [11, 67]. Apesar de utilizarem diferentes nomes na identificação do tipo de acto comunicativo são bastante idênticas nos seus conceitos e princípios básicos.

As ontologias dão o seu contributo ao definirem o vocabulário que será usado na comunicação entre agentes e o conhecimento associado a cada um desses termos, onde se inclui a definição dos conceitos e suas relações. De forma tradicional a informação encontra-se numa linguagem lógica que torne possível a distinção de classes, instancias, propriedades, relações e funções de forma clara e consistente. O propósito das ontologias é criar um entendimento partilhado entre entidades cooperativas, de modo a permitir a troca, armazenamento, reutilização e verificação de consistência dessa informação [77].

Por fim uma cooperação ou colaboração depende do êxito na distribuição das tarefas constituintes da acção original. Tal operação pode ocorrer por um processo de contratação onde um dos protocolos mais comum é o Protocolo Contratual de Rede - CNP[13] [78], baseado no mecanismo utilizado por empresas na troca de serviços e bens. O seu funcionamento resume-se ao envio de um pedido no qual se descreve a necessidade de realizar determinada operação. Os agentes habilitados e disponíveis a realizar essa operação respondem a quem os interpelou. Após análise das respostas, quem apresentar a alternativa mais adequada ganha o contrato.

Normalmente associados à manufactura pela área industrial também provam o seu valor na integração inter-empresarial. Devido à globalização as empresas tendem a dividir-se em pequenas sub-empresas, pertencentes ou não à mãe, e especializadas num determinado tipo produto. A expansão geográfica, por distribuição das diversas fábricas de produção, fornecedores, área administrativa e comercial eleva a troca de informação, coordenação, tomada de decisões e monitorização a outro nível [79]. Neste novo tipo de organização, as empresas deixam de ser vistas como entidades individuais, forçando a rever a forma como estão organizadas e a introdução num novo conceito – empresas virtuais [80, 81]. A aplicação de agentes na integração empresarial incute-lhes as suas características de fiabilidade e robustez ao permitir a monitorização e continuidade das operações mesmo na presença de falhas de comunicação. A integração, apesar de ser um problema

[12] Knowledge Query and Manipulation Language
[13] Contract Net Protocol

antigo, está longe de ser resolvida. Restrições entre as diversas áreas, questões temporais, de escalabilidade e ao nível da representação de modelos necessitam de análise.

Com a complexidade dos sistemas a aumentar, há necessidade de uma maior sofisticação nos sistemas de controlo o que por seu lado requer abstracções e metáforas mais poderosas de modo a conceber e explicar o seu funcionamento. Os MAS ajudam nesse processo ao permitirem a estruturação do conhecimento em torno de componentes autónomos e com capacidades comunicativas.

Existe um grande número de projectos em desenvolvimento que almejam oferecer todas as características atribuídas a estes sistemas e entidades. Plataformas como o JADE[65], ZEUS[82], Cougaar [83] , são algumas das alternativas disponíveis, contudo nenhuma delas é verdadeiramente distribuída pois na sua grande maioria dão ênfase à comunicação e modelação de agentes.

Em resumo, as aplicações de MAS na indústria encontram-se maioritariamente em três áreas [27]: controlo de manufactura em tempo real; gestão de produção onde se inclui o planeamento, agendamento, pedidos, pré processamento; e empresas virtuais que integram a manufactura, rede de vendedores, fornecedores, distribuidores [84]. A grande diversidade de áreas onde a sua aplicação é vista como vantajosa e as diversas aproximações possíveis deixam antever o potencial da tecnologia.

Actualmente os agentes e as sociedades que se constroem em torno dos mesmos são tidos como uma solução adequada aos requisitos de autonomia, interacção, comunicação e cooperação dos sistemas de manufactura actuais.

Apesar de as perspectivas serem positivas são ainda essenciais provas significativas da aplicabilidade e vantagens destes paradigmas nos ambientes de manufactura. Tal fica a dever-se ao número diminuto de aplicações expostas [27]. As implementações mais conhecidas são um sistema de controlo de produção para a DaimlerChrysler [85] e um sistema de controlo de circulação de águas destinado a arrefecer os equipamentos nos Navios dos EUA [86]. Mais aplicações podem ser encontradas em [87].

Entre as diversas razões existentes, algumas das quais já indicadas, é possível acrescentar [27, 87-90]:

- A introdução de agentes, em princípio, não irá reduzir a complexidade dos sistemas. O aspecto anterior tem como exemplo de causa o nível de interacções envolvidas;
- Interoperabilidade quer entre diferentes plataformas quer entre os agentes e dispositivos antigos é dispendiosa;
- Impossibilidade da plataforma em lidar com elevado número de agentes, que ao contrário das dezenas ou centenas dos protótipos desenvolvidos, se situa na casa dos milhares;

- Ausência de tecnologias, ferramentas e *hardware* que auxiliem no desenvolvimento e suportem a execução de agentes;
- Inexistência de especificações e metodologias de simulação, desenvolvimento, teste e validação dos requisitos das arquitecturas e na integração com sistemas antigos;
- Suporte dos sistemas possível de ser efectuado apenas pelos construtores e vendedores, pois são os únicos que compreendem e conhecem as nuances dos sistemas;

Independentemente de todas as limitações, um dos principais entraves continua a ser a característica emergência devido à incerteza a ela associada. A indústria quer total previsibilidade e garantias a fim de conseguir fiabilidade e *performance* operacional.

Os agentes como todas as tecnologias novas trazem consigo alguma incerteza, mas de uma maneira ou outra irão fazer parte dos sistemas de manufactura futuros.

2.5 Regras e Fluxogramas em Agentes

Reengenharia de processos é o elemento chave para as companhias recuperarem a competitividade e lucro nos mercados cada vez mais voláteis. A adição de sistemas emuladores de raciocínio e fluxogramas irá beneficiar os agentes nos aspectos de socialização, adaptabilidade e reconfigurabilidade que em algumas situações não recebem a devida atenção, mas com importância no processo de reestruturação [91, 92].

Relativamente aos WfMS, a tecnologia de agentes tem sido utilizada das mais diversas formas e há quem defenda que será o mecanismo de *enactment* do futuro [93, 94]. Em algumas situações os agentes encontram-se integrados no WfMS [95, 96]. Esta alternativa marcada pela modularidade dos diversos componentes, permite expor as funcionalidades e vantagens associadas aos agentes e assim oferecer flexibilidade superior com relação aos sistemas comuns.

Noutros casos, mais semelhantes ao sistema proposto, os agentes desempenham determinadas funções requeridas pelas tarefas descritas no fluxograma. Nestes casos, o fluxograma é usado na modelação e posterior coordenação dos agentes [97, 98]. Lamentavelmente grande parte das alternativas apresentadas encontra-se na área de negócio B2B[14] [99, 100] e provavelmente por esse motivo não descem à especificação de cada um dos sub-processos representativos das actividades pertencentes ao processo principal. As soluções mais próximas do sistema que se irá apresentar são dadas por Yanli [101] onde conjuntamente com uma arquitectura similar existe um gestor de trabalho destinado a reduzir os tempos de *stanby* das máquinas. Nesta alternativa, a atribuição de tarefas a

[14] Business to Business

recursos é efectuada de forma dinâmica, os agentes puxam as tarefas (em vez de lhe serem empurradas) e a descrição do processo encontra-se ao nível global. No Jbees [102] é mantida a atribuição dinâmica e a distribuição de actividades pelos agentes envolvidos, contudo a definição de processos ocorre através de redes de petri coloridas e são utilizados *webservices* para encapsular a comunicação com os recursos. Uma referência ao projecto ADACHOR que também menciona a existência de um sistema de planeamento, este também destinado à sequencia global das operações. Por outro lado, existe o projecto WASA [103] onde já se aufere o conceito de sub-processos. A flexibilidade por ele oferecida baseia-se no conceito de meta-actividades.

Uma outra implementação a merecer destaque é dada pelo sistema GAIN [104] baseado na arquitectura GRID, sobre o qual tem sido realizada diversa investigação quer ao nível da utilização de agentes [105, 106] quer ao nível dos fluxogramas [107, 108]. Este sistema, por exemplo, baseia-se nas ideias de Buhler[94] que propõe o uso de agentes para composição dinâmica de *webservices* através de fluxogramas e do *GridFlow* [109] onde os agentes são utilizados para gestão dos recursos GRID. Tal como na implementação que se irá expor, recorre-se ao JADE nos aspectos relativos aos agentes e XPDL para a definição de processos. De igual modo é suportado o conceito de sub-processo mas num contexto diferente. A grande vantagem referida prende-se com o facto de permitir a replicação de trabalho e a substituição de uma tarefa falhada por outra ou outras que obtenham o mesmo objectivo. Situação também prevista no software do autor.

Por fim, projectos como o Jboss [110] ou o JADE [65] começam agora a apresentar soluções onde se recorre a fluxogramas. Estes, apesar de permitirem a execução de fluxogramas, apresentam limitações relativas à flexibilidade nomeadamente nos aspectos relativos a modificações dinâmicas.

Nos aspectos relativos ao aproveitamento de sistemas baseados em regras para utilização em ambientes multiagente, a sua aplicabilidade mais directa relaciona-se com a emulação do raciocínio [111, 112]. Devido a essa capacidade os agentes conseguem eficientemente detectar lacunas, providenciar aconselhamento ou tomar decisões sempre com consideração ao *feedback* recebido. O processamento de objectivos complexos onde se inclui o processamento paralelo e trabalho de equipa também é suportado.

Por vezes o pretendido é que o sistema reaja a determinados acontecimentos. Nessas situações existem implementações, como o SARI, capaz de observar e avaliar eventos à procura de padrões pré determinados com intuito de conseguir responder atempadamente a alterações no ambiente. O SARI[113] inova ao introduzir a correlação de eventos, importante quando a ordem dum grupo de ocorrências é importante.

Outra área sobre a qual tem recaído a investigação está relacionada com a construção de sequências de montagem. Apesar de cada implementação apresentar as suas particularidades, o

princípio básico mantêm-se comum à maioria das alternativas: decompor uma operação num conjunto ordenado de actividades mais simples e directamente realizáveis por entidades físicas. Veja-se o exemplo de Botea [114] e Swaminathan [115] onde se introduzem duas dessas situações.

3 Tecnologias de Suporte

3.1 Sistemas Baseados em Regras

3.1.1 Introdução

O processo de raciocínio sempre intrigou filósofos e cientistas e muitas teorias foram levantadas sobre esse assunto.

Com a evolução surgiram os computadores e as linguagens de programação e desde essa altura que se tem tentado reproduzir o processo de raciocínio humano. De modo a agrupar todo o conhecimento relativo a essa área e facilitar o planeamento e distribuição da investigação, surge uma nova área denominada de inteligência artificial - AI. De entre as diversas ramificações, os sistemas periciais - *expert systems* - são dos que têm recebido mais atenção. Dos diversos elementos constituintes, como seja o de obtenção e representação do conhecimento ou explicação de soluções, o de maior relevo está associado à ciência cognitiva ou raciocínio [116]. Cognição é o estudo de como os humanos processam a informação [117]. Por outras palavras, é o estudo de como as pessoas pensam, especialmente ao resolver problemas [118]. A sua importância fica demonstrada quando se pretende simular o conhecimento de peritos e estes por seu lado não conseguem explicar como resolveram um problema.

Por Graham [119] deduz-se facilmente a ausência de compreensão nos aspectos relativos ao funcionamento do cérebro humano e a falta de concordância/aceitação no que diz respeito aos formalismos para representação do conhecimento. Apesar de toda a controvérsia que envolve este tema, as regras são vistas como um boa opção de representação. A consideração anterior aliada ao facto de funcionarem com base nesse mesmo conceito, elevam os sistemas baseados em regras a método de eleição na construção de sistemas periciais, destinados a resolver problemas [120].

Em [121] Friedman-Hill define uma regra como "*um tipo de instrução ou comando aplicável em determinadas situações*". Segundo o autor, tal descrição leva à ideia de que todo o conhecimento sobre o mundo pode ser codificado sobre a forma de regras. A experiência demonstra que apesar de essa ocorrência ser a mais comum, existem diversas situações onde a mesma não se verifica.

Numa primeira observação poder-se-ia questionar a utilidade deste paradigma na actualidade, mas uma análise mais pormenorizada mostra a sua integração em praticamente todos os negócios e casas [121], muitas vezes sem o utilizador se dar conta. Basta considerar o exemplo do escritório moderno onde muitas actividades são controladas por regras como é o caso dos sistemas de e-mail e os vários tipos de filtragem oferecidos, ou a tradução de endereços (Outlook, Eudora); As vendas na internet, com recomendação de produtos semelhantes; As aplicações de diagnóstico médico, encarregues de apresentar possíveis efemeridades de que padecem os doentes. Os sistemas de

configuração de produtos, que escolhem os componentes e elementos necessários para as características pedidas por um cliente - XCON[15]. Este último e o seu predecessor XSEL, destinado à empresa DEC[16], são sistemas de referência pois trouxeram grandes poupanças provenientes do aumento de precisão, redução dos custos de teste e aumento da satisfação do cliente.

As vantagens estendem-se também a nível empresarial, onde a separação explícita entre as estratégias e o código permite a modificação sem ser necessário recorrer a programação. Este e outros factores juntos com a possível utilização de aplicações genéricas diminuem os custos e o tempo de resposta a mudanças [121].

Muita atenção tem sido dada ao tópico de codificação do conhecimento, nomeadamente em termos da utilização de "linguagens naturais". Junto com a adaptação ao domínio, a natureza da tarefa em mãos, a disponibilidade e a quantidade de informação necessária ao eficiente processo de inferência são alguns dos aspectos determinantes no sucesso do sistema no ambiente actual de agressiva competitividade e constante mudança [116]. Normalmente, neste tipo de sistemas e contrariamente aos procedimentais, onde se utilizam algoritmos, recorre-se a regras no formato condicional na representação do conhecimento humano, utilizado posteriormente conforme for requerido no processo de raciocínio.

Devido ao facto de o conhecimento se encontrar na forma modular, os sistemas baseados em regras podem "degradar-se graciosamente" na presença de informação incompleta. Por Friedman-Hill [121] "*a qualidade não se constrói, está lá!*", expressão possível de ser traduzida em frases como: Quando o número de variáveis é elevado o sistema fornece todas as alternativas.

Os componentes básicos de um sistema deste tipo são a base de conhecimento responsável por guardar toda a informação, dados e regras; o motor de inferência que procura relações entre o conhecimento e os factos; o mecanismo de explicação que permite ao utilizador compreender como o sistema chegou àquela conclusão; o mecanismo de aquisição de conhecimento encarregue de facilitar a introdução de conhecimento no sistema; a interface de utilizador destinada a permitir a comunicação entre máquina e utilizador.

Apesar de ainda longe, prevê-se que algum dia, sistemas periciais sofisticados sejam capazes de reproduzir a inteligência humana e as habilidades de resolução de problemas [121].

[15] e**X**pert **CON**figurer, também conhecido como R1
[16] **D**igital **E**quipment **C**orporation

3.1.2 Paradigmas da Programação

Este capítulo situa os sistemas baseados em regras por entre as linguagens de programação mais comuns. Tal como é apresentado na Figura 3.1, os paradigmas de programação podem ser classificados como procedimentais e não procedimentais.

Nos primeiros, também designados de sequenciais, tem-se as linguagens imperativas e funcionais, nas quais o programador tem de especificar como a solução deve ser obtida. Nos não procedimentais, onde se inclui a programação orientada a objectos, lógica, e as máquinas de regras, pretende-se por outro lado apenas a especificação do objectivo ou solução a atingir

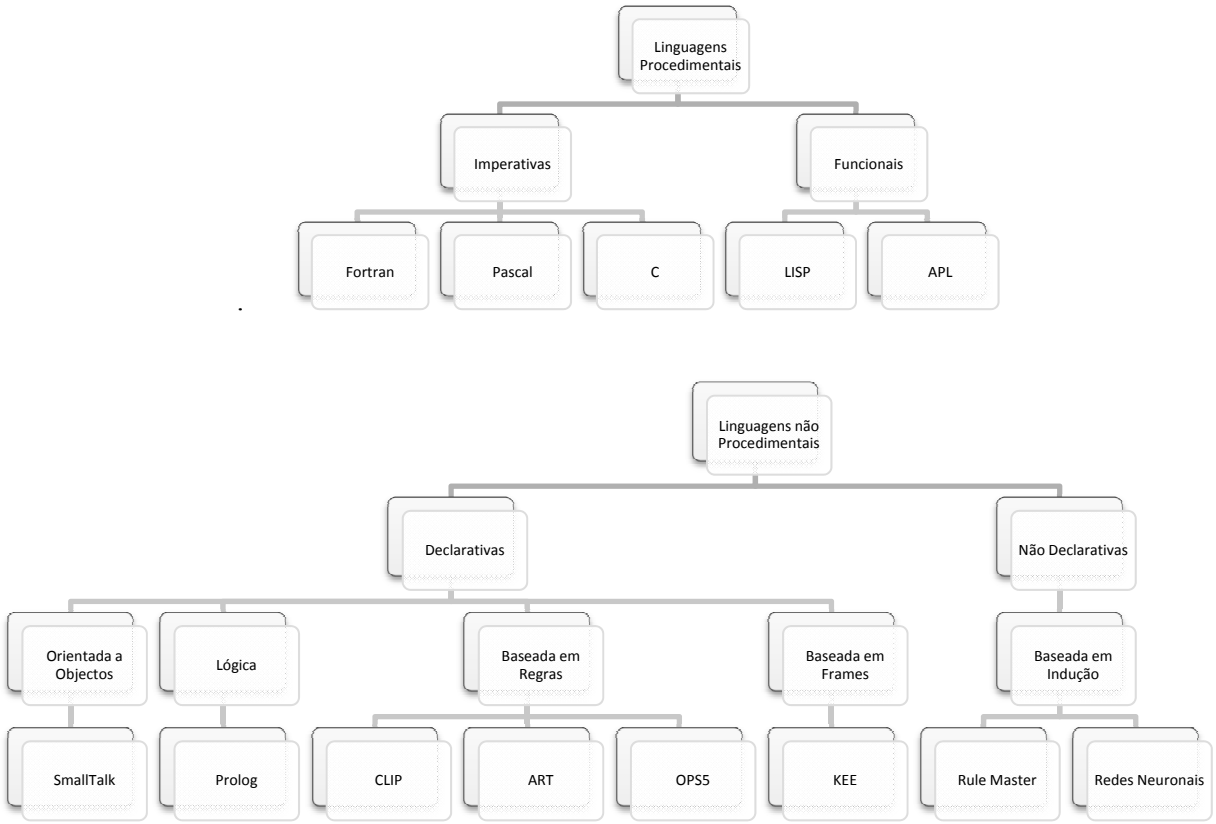

Figura 3.1 - Paradigmas Programação [118].

Por Riley[118] os Sistemas Periciais dos quais fazem parte os sistemas baseados em regras encontram-se no subgrupo das linguagens declarativas. Este tipo de programação caracteriza-se pela separação entre objectivo, dos métodos para o atingir. Nele, o utilizador especifica o objectivo mas a decisão da sequência de acções fica a cargo sistema.

Como seria de esperar, as diferenças entre a programação convencional e os sistemas periciais são muitas. A merecer especial destaque está o facto de o controlo ser efectuado pela máquina de inferência, haver uma separação explícita entre lógica e código, a solução ser dada por regras e de ser fornecida uma explicação de como se chegou à conclusão. Nenhuma das características anteriores se

encontra presente na programação convencional onde não é oferecida explicação do resultado obtido, a execução é feita pela ordem das expressões, o conhecimento aparece embebido no programa e a solução é fornecida através de algoritmos.

A capacidade de simular o raciocínio torna estes sistemas no método de eleição em situações onde existe incerteza. Uma vez que terão de lidar com dados incorrectos, incompletos ou inconsistentes na informação introduzida, as suas respostas irão depender do conhecimento adquirido, situação impossível de alcançar com os algoritmos.

3.1.3 Definição

A posterior análise das características, funcionalidades, componentes e propriedades destes sistemas tem de ser precedida pela interiorização do conceito de máquina de regras. Motivada pelas diversas áreas abrangidas e pelo elevado número de aplicações existe uma grande variedade de definições, cada uma com o seu próprio foco.

A título ilustrativo apresentam-se agora algumas alternativas, seguidas por uma definição geral do autor.

(1) Por Graham[119] os Sistemas Baseados em Regras também designados por vezes de bases de conhecimento (knowledge-based systems), são sistemas de computadores capazes de fornecer opiniões ou concelhos e tomar decisões numa área bem definida ao nível do perito humano. Segundo este autor existem dois grandes tipos, os que apenas aconselham acções a tomar, denominados de sistemas de suporte ou confirmação, e os que realizam acções ou tomam decisões.

(2) Pelo professor Edward Feigenbaum da Universidade de Stanford, um Sistema pericial é definido como um "*programa de computador que utiliza o conhecimento e procedimentos de inferência na resolução de problemas onde é necessária uma quantidade significativa de perícia humana na procura de soluções*" [122].

(3) Sistema pericial é um programa de computador que representa e usa as capacidades do conhecimento de um ou mais peritos no fornecimento de *performance* de alta qualidade num domínio específico. Tais características são atingidas através de um programa que emula a habilidade de decisão de um perito[123].

Segundo o autor, é um *software* emulador do processo de raciocínio. Utiliza o conhecimento previamente introduzido por peritos na apresentação de soluções, sobre a forma de conselhos ou acções, difíceis de obter por indivíduos comuns sem experiencia. Logo, entre outras estruturas, necessita de uma área de armazenamento de informação, onde se encontram as regras definidas na forma declarativa.

3.1.4 Importância destes Sistemas

As práticas correntes de desenvolvimento de *software* possuem diversos problemas ao nível da replicação e modificação de código, tempos e custos de manutenção. Estas e outras limitações começaram a levantar questões sobre a sua viabilidade em determinadas áreas [119]:

- Como desagregar de forma eficiente a parte lógica da programação de modo a ser possível proceder a alterações sem haver necessidade de refazer o código?
- Como possibilitar a sua utilização em várias aplicações? Como facilitar uma utilização multi-cultural?
- Como possibilitar a utilização de múltiplas linguagens na representação da parte lógica do programa?
- Como colmatar a falta de um perito com conhecimentos de programação que exponha e codifique, durante o desenvolvimento, qual a lógica de programa necessária?
- Como permitir a personalização de serviços, conteúdos e interacções com base nas características do cliente de forma a adicionar valor à empresa;
- Como partilhar a informação de forma coerente e consistente;

Estes e outros pontos permitem-nos concluir sobre a necessidade de um novo método de programação quando se pretende agilidade e flexibilidade. O desenvolvimento de aplicações com o objectivo de solucionar as questões anteriores, já decorre há mais de 30 anos. Durante esse tempo foram sendo definidas as características que deveriam ser expostas [116, 119]:

- Permitir a verificação da sintaxe e lógica implementada;
- Facilitar a criação de várias regras semelhantes com recurso a *templates*;
- Permitir a rápida alteração das regras quer seja pelo requisito de um novo produto quer pela necessidade de oferecer personalização. Só assim se consegue oferecer competitividade;
- Permitir a invocação de procedimentos por regras e vice-versa;
- Fornecer alternativas ao método IF…THEN, de definição de regras. Como sejam as árvores de decisão, úteis em depuração, ou as tabelas de decisão, onde a informação se apresenta num formato tabular;
- Permitir a reutilização de regras entre aplicações;
- Possibilitar a escalabilidade do ambiente e garantir funcionamento em situações com limitação de tempo ou de elevada actividade;
- Capacidade em lidar com conhecimento impreciso, incerto e/ou difuso e aprender através de exemplos e experiencias passadas, de modo a aumentar a eficiência perante problemas novos e recorrentes;

- Conseguir explicar o seu processo de raciocínio, justificar a conclusão apresentada e responder a questões sobre o processo de inferência;

Através da lista de requisitos é possível antever quais as propriedades necessárias neste novo tipo de *software*, algumas das quais se apresentam no subcapítulo seguinte.

3.1.5 Propriedades destes Sistemas

Apesar de haver diversas formas de organizar o conhecimento nos sistemas periciais, os sistemas baseados em regras têm algumas particularidades próprias [116, 120]:

- Incorporam conhecimento heurístico em regras condicionais do tipo IF…THEN e utilizam esse conhecimento na simulação do raciocínio com o intuito de obter uma solução que, todavia, pode não ser a melhor;
- As suas capacidades e perícia são proporcionais ao tamanho da sua base de conhecimento (regras);
- Conseguem resolver um vasto leque de problemas complexos, pelo seleccionar das regras de interesse, em conjunto com a sequência de aplicação.
- Explicam as decisões tomadas, traduzindo as regras utilizadas para linguagem natural. Permitem, assim, satisfazer as necessidades de compreensão por parte dos utilizadores.

As aplicações resultantes oferecem, deste modo, capacidades únicas aos seus utilizadores.

3.1.6 Vantagens

Os Sistemas Baseados em Regras surgiram como solução de integração entre a parte lógica e o código em si. Têm como principais vantagens [116, 118, 120, 124, 125]:

- A propriedade de isolamento entre código e regras é benéfica a vários níveis:
    - Possibilita um elevado nível de independência face a linguagens de programação. Consegue-se assim um funcionamento semelhante à programação por objectos ou módulos, na qual, quando a implementação de uma estrutura ou função se altera, essas modificações não se propagam a outros objectos;
    - Maior paralelismo e modularidade, pois não é necessário definir ordem ou dependências entre regras, como acontece na programação por procedimentos. A máquina é quem decide quando avaliar e/ou activar cada regra, baseando-se na informação por ela necessária e em conclusões anteriores;
    - Produção de programas mais pequenos, em comparação com outras linguagens, mais fáceis de construir, alterar e testar;

    - Maior agilidade, ao possibilitar a rápida alteração ou adição de regras, sem ser necessário recorrer à reprogramação. Essa característica tem como objectivos mais visíveis o aumento da *performance* ou a realização de testes;
    - Programadores podem assim empenhar-se na sua real função – construção do programa - e deixar a definição do conhecimento junto com os aspectos relativos a como proceder à sua representação, para os peritos em conhecimento da área;
    - Reutilização de regras, com o intuito de serem distribuídas por vários sistemas;
- Permitir distribuir e disponibilizar o conhecimento quando o número de peritos é reduzido e em situações onde existe risco de vida. Faculta-se, assim, perícia a entidades menos instruídas;
- Possuir mecanismos de aprendizagem através de exemplos e experiências passadas, que possibilitam a criação de novas regras;
- Não exigirem elevado nível de processamento, apesar de requerer alguma quantidade de memória, utilizada no armazenamento de todo o conhecimento;
- Desde que preparados, conseguem lidar com incerteza, dados incompletos, incoerências e fornecer um factor de confiança;
- Devido ao conhecimento adquirido ser permanente oferecem consistência superior a um perito humano, cuja perícia é afectada por diversos factores;
- De forma a aumentar ainda mais a sua eficiência e taxa de sucesso pode combinar o conhecimento de vários peritos. Consegue-se desta forma, num elevado número de vezes, superar o humano em velocidade e capacidade;
- As suas técnicas de resolução de problemas podem ser usadas numa grande variedade de aplicações;

Em adição aos factores enunciados, o seu baixo custo de elaboração e manutenção aliado à capacidade de explicação do processo de raciocínio, fazem deste, um sistema de eleição.

3.1.7 Desvantagens e Limitações

Por ser uma tecnologia relativamente recente, durante algum tempo, ainda necessitará de aperfeiçoamentos e afinações de modo a suprimir as limitações agora expostas [116, 119-121]:

- Como dito por Giaratano e Riley "*a não ser que os sistemas estejam preparados para lidar com incerteza, informação incompleta ou imprecisa, irão apresentar as suas recomendações com confiança equivalente*";
- Ausência de conhecimento causal, pois não tem real conhecimento das causas, dos factos, e efeitos, das suas acções ou recomendações. Apenas possui conhecimento empírico e heurístico.

- A sua perícia é limitada à área onde irá ser aplicado. Ao contrário dos humanos não consegue generalizar o seu conhecimento recorrendo a analogia, possuir senso comum e criatividade na resposta a questões fora do habitual;
- Interoperabilidade limitada, devido à ausência de *standard* na linguagem de representação das regras. Causa do aparecimento de múltiplas aplicações divergentes apenas na linguagem, ajustada à área de destino;

Algumas soluções propostas passam pela instrução de utilizadores em linguagens de computador, desenvolvimento de linguagens naturais e disponibilização de interfaces intuitivas de gestão de regras;

- Apesar de já existirem aplicações a facultar acesso remoto, ainda serão necessárias modificações e actualizações até que a integração num ambiente distribuído seja conseguida de forma eficiente.
- A ausência de métodos ou técnicas de verificação destinados a testar de forma exacta a coerência e o nível de conhecimento de um bloco de regras, nomeadamente nos aspectos relativos à existência de regras contraditórias.
- Os requisitos elevados de memória podem elevar o seu grau de ineficiência. A sua velocidade irá depender, entre outros factores, da velocidade de acesso à informação e do algoritmo utilizado. Na actualidade, de forma a não diminuir a eficiência, o conhecimento é organizado em módulos ou grupos de acordo com a sua área de aplicação.
- Algumas linguagens utilizadas podem requerer compiladores específicos.

Estas restrições levam à representação do conhecimento segundo métodos que conjugam regras, objectos e procedimentos.

3.1.8 Fases de Construção

Normalmente o desenvolvimento de um sistema pericial inclui as seguintes fases: definição do conceito; estudo de realização; descrição geral das especificações; aquisição preliminar de conhecimento; selecção de ferramentas; representação do conhecimento; desenvolvimento de um protótipo, onde o sistema é avaliado pelo cliente; aquisição principal de conhecimento; revisão da especificação; desenvolvimento do sistema; teste; avaliação e entrega. A referir ainda o facto de o circuito de desenvolvimento indicando dispor em algumas fases de realimentação [126]. Nos subcapítulos seguintes descrevem-se as fases consideradas de maior importância para a implementação e posterior integração do sistema.

3.1.8.1 Desenvolvimento do Sistema

Linguagens de programação e *expert system building environments* são algumas das diversas ferramentas destinadas à construção de sistemas periciais. Se o objectivo for elaborar um sistema de raiz recorre-se a linguagens especializadas. As de maior expressão são: LISP, PROLOG, OPS-5, SMALLTALK, ou, noutro extremo, o C++ que permite uma grande compatibilidade, velocidade e portabilidade. A introdução da linguagem Java é recente mas está a ganhar grande popularidade, devido à portabilidade entre diferentes plataformas computacionais.

Outra opção, caracterizada pela maior simplicidade, é dada pelos ambientes de desenvolvimento de sistemas periciais KAS, OPS5, AGE. Aqui disponibiliza-se um grande número de capacidades e opções. É possível, por exemplo, escolher qual o formato de representação do conhecimento e qual o mecanismo de inferência a utilizar [126].

3.1.8.2 Obtenção e Angariação de Conhecimento:

O próximo passo na construção prende-se com o "extrair", estruturar e organizar do conhecimento de domínio proveniente do perito.

Devido aos requisitos de tempo e complexidade envolvidos, as etapas enunciadas tornam-se muitas vezes o ponto de congestão no desenvolvimento destes sistemas. Na tentativa de automatizar estes processos e resolver os problemas expostos, recorre-se a ambientes orientados ao domínio com métodos específicos de representação do conhecimento (OPAL, PROTEGE e SALT). As aplicações resultantes eliminam a necessidade de engenheiros de conhecimento e reduzem o número de ajustes a efectuar pelo método cíclico, visto possibilitarem a introdução do conhecimento pelo utilizador final. É, no entanto, necessário ter em consideração a ocorrência de erros devido ao provável conhecimento limitado que o utilizador dispõe das diversas aplicações.

Quando não é utilizada a alternativa anterior, o conhecimento tem de ser adquirido com recurso a peritos, métodos teóricos e/ou de indução. Este último método é especialmente útil quando se está perante um número elevado de casos, não existe um perito na área ou se pretende que o sistema aprenda a dar mais importância a determinadas regras [116].

O processo de adquirir conhecimento funciona de forma cíclica e decorre alternadamente com o de representação até se obter um desempenho satisfatório. Infelizmente, e ao contrario das convicções comuns, a obtenção de conhecimento não ocorre de forma directa. Associado a ele existe uma análise e filtragem profunda de informação.

3.1.8.3 Representação Conhecimento

O conceito de conhecimento requer mais que as convencionais estruturas usadas em bases de dados. Existem vários tipos, reflectidos nas diversas formas de representação existentes. Essa multiplicidade é distinguível pela própria representação utilizada, pelo seu nível de processamento,

pela especificidade ao domínio, precisão e incerteza [126]. A escolha de regras, objectos ou procedimentos para a sua representação, irá afectar o sistema na sua eficiência e rapidez de manutenção [127].

Ao nível das regras, o formato e sintaxe variam consoante a máquina de inferência e as ferramentas utilizadas na sua implementação e manutenção. De forma comum, são utilizadas expressões condicionais, denominadas também de situação-acção ou regras de produção, úteis na representação de heurísticas de como um perito faz as coisas ou o que faz sem saber o porquê. Em alternativa, é possível recorrer às redes semânticas e frames, árvores de decisão, tabelas de decisão e linguagens lógicas.

3.1.8.4 Regras – O Elemento Constituinte

Mais uma vez, a grande variedade de áreas que utilizam regras resulta numa pluralidade de definições, cada uma mais orientada ao domínio de destino. De forma geral, uma regra pode ser definida como: "*Uma condição lógica com informação contida no formato de comando ou instrução. Constituída no mínimo por uma condição e uma acção ou conclusão, é utilizada pela máquina de inferência de forma a solucionar o problema apresentado*". Contudo, a sua natureza leva-a a ser aplicável apenas em determinadas situações e a informar unicamente sobre o acto a realizar mas não do "como" [121].

Apresentam normalmente, uma estrutura do tipo SE condição ENTÃO conclusão onde a secção entre o "SE" e o "ENTÃO", denominada de antecedente, condicional, padrão, predicado, premissa ou *left hand side* - LHS é interpretada da seguinte forma: se a ou as condições forem satisfeitas podemos inferir uma nova expressão ou uma determinada operação. A secção depois do "ENTÃO", intitulada consequente ou *right hand side* - RHS contém a série de acções a realizar.

O modelo de representação utilizado permite exprimir *conhecimento dedutivo*, como por exemplo as relações lógicas, que desta forma suportam inferências e tarefas de avaliação; *conhecimento orientado a objectivos*, utilizado na procura de soluções e justificações do comportamento tomado; *relações causais*, de interesse em questões do tipo "então e se" ou na determinação de causas especificas de eventos;

Pode-se então concluir que as vantagens deste modelo encontram-se na [119, 120, 124, 125, 127]:

- Natureza modular, declarativa e independente – permite o fácil encapsulamento de informação, teste, manutenção e a progressiva expansão do sistema.
- Facilidade com que permitem construir o componente de explicação – pois as condições podem também ser vistas como as causas de acções.

- Similaridade ao processo cognitivo humano – baseado na teoria de Newell e Simon [128], as regras são o mecanismo natural de modelar como os humanos resolvem os problemas, ou seja, o seu conhecimento do que os rodeia.
- Estrutura simples, causa-efeito ou situação-acção – Facilita a explicação da estrutura pretendida no conhecimento, a sua representação pelo perito e a compreensão por parte dos computadores.

3.1.9 Arquitectura Comum

Os sistemas periciais podem funcionar a título individual ou como um assistente. Na segunda situação, o utilizador começa por fornecer factos e em troca recebe aconselhamento profissional em termos de possíveis resultados, soluções e acções a tomar.

A Figura 3.2 mostra os principais elementos de um sistema baseado em regras:

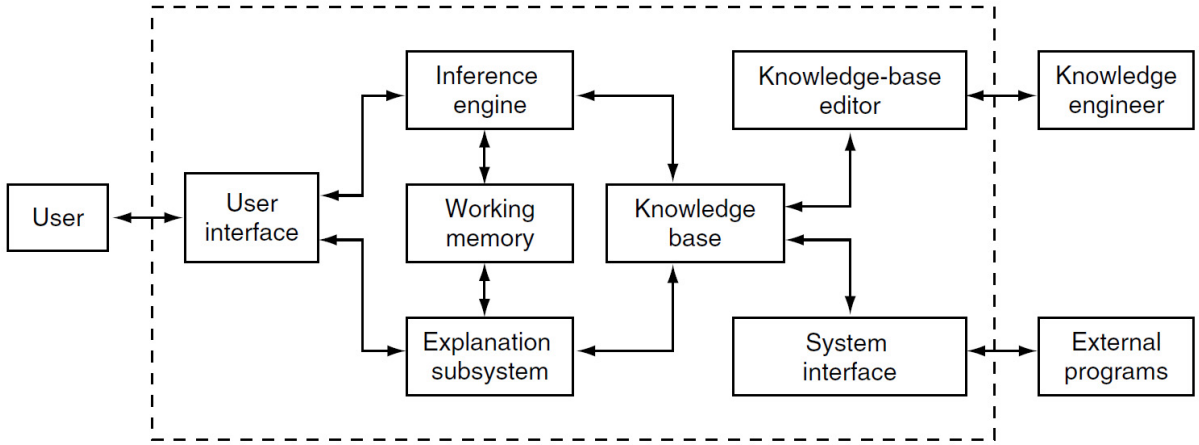

Figura 3.2 - Arquitectura tipica de um Sistema pericial [123]

Como principais elementos diferenciadores da programação convencional, os RBS efectuam a procura de padrões entre regras e factos, e conseguem identificar rapidamente as regras relevantes à medida que a memória de trabalho se altera [120].

O número de módulos apresentados por um sistema deste tipo varia consoante as capacidades pretendidas para o sistema final. No entanto, os elementos considerados fundamentais são agora expostos.

3.1.9.1 Base de Conhecimento (Memória de Produção)

Utilizada pelo motor de inferência na obtenção de conclusões, contém informação especializada do domínio, normalmente resultante da combinação entre conhecimento teórico e de vários peritos, codificada segundo as alternativas de capítulos anteriores.

Com o propósito de aumentar a eficiência muitos sistemas implementam metaregras, adicionam informações de classificação e possuem mecanismos destinados a organizar as premissas ou conclusões das regras [121].

3.1.9.2 Interfaces

De forma comum são disponibilizadas três interfaces. A de sistema destina-se a possibilitar a conexão a programas externos ou fontes de informação adicionais, como sejam outros sistemas periciais. Por outro lado, através da interface de utilizador é possibilitada a interacção com a aplicação. Torna-se, deste modo, o sistema útil e fica facilitada a utilização em tarefas de fornecimento de informação e recepção de respostas [124, 126]. Por último, a interface do perito destina-se ao fornecimento de conhecimento que permitirá ao sistema apresentar soluções.

Devido ao requisito de intuitividade, as diferentes interfaces disponibilizam mecanismos como o das questões e respostas, orientação ao menu, linguagem natural, gráficos e ajuda online de modo a simplificar o processo de interacção.

3.1.9.3 Mecanismo de Tradução

Apesar de não se encontrar presente permite a representação de regras em várias linguagens e/ou formatos, com intuito de facilitar a introdução, interpretação ou explicação dos dados. Normalmente, todas as regras são mantidas num formato e traduzidas quando necessário.

3.1.9.4 Mecanismo de Explicação

Considerado essencial por Dhaliwal e Benbasat, Ye e Johnson [116], disponibiliza a capacidade de explorar alternativas do tipo "então e se", também denominadas de raciocínio hipotético, e permite clarificar a tomada de decisões ou recomendações fornecidas.

Dos diversos tipos de explicação existentes o "porquê" e o "como", introduzidas de forma pioneira no MYCIN [129], são os mais frequentes e continuam a ser os elementos principais dos mecanismos de explicação de hoje. O primeiro fornece conhecimento explicativo das razões que levam a uma acção, baseada no modelo causal. No segundo, apelidado de *Rule Tracing*, é mantido um registo do caminho percorrido até se chegar à conclusão apresentada, utilizado na posterior apresentação da linha de raciocínio.

3.1.9.5 Memória de Trabalho

Tem como função, guardar a informação actual sobre a qual o sistema baseado em regras se encontra a trabalhar e consoante as necessidades, recuperar regras da base de conhecimento. A memória de trabalho é a base de dados dos factos, conjugados pelo motor de inferência com as regras. Armazena também as decisões, hipóteses e resultados parciais, particularmente a informação sobre o estado de resolução do problema [116, 120]. Dispõe também de uma agenda que contém e organiza as regras habilitadas para execução, auxiliando na resolução de potenciais conflitos.

3.1.9.6 Motor de Inferência

Destina-se a procurar relações com recurso à base de conhecimento, controlar todo o processo de aplicação de regras a dados e fornecer respostas e sugestões de maneira semelhante à de um perito humano [121, 124]. Logo durante o seu funcionamento discreto, tem de decidir quais regras são

satisfeitas pelos factos, organizá-las por prioridades – resolução de conflitos - e, com recurso à agenda, executar a regra com prioridade mais elevada. Assim, a regra executada poderá, por seu lado, alterar condições externas ou em alternativa modificar ou adicionar elementos à memória de trabalho, dando início a um novo ciclo.

De entre os diversos algoritmos de inferência disponíveis, *means-ends analysis*, *problem reduction*, *plan generate and test*, *hierarchical planning*, *commitment principle*, *constrain handling*, *depth-first search*, *breadth-first search*, *hill climbing*, *pattern matching,* todos eles têm por base um mecanismo de inferência directo ou inverso, também denominados de *forward chaining* e *backward chaining*, ou estratégias gerais de raciocínio.

O *forward chaining* ou orientado a estímulos, parte dos factos para as conclusões que dele possam resultar. Segundo Christoph[130] parte de factos, dados como conhecidos, e tenta de forma sucessiva, aplicar todas as regras possíveis com o propósito de descobrir o maior número de novas sub-conclusões realizáveis. Nesta estratégia, uma regra é disparada quando ocorrem alterações na memória de trabalho que coincidem com o antecedente da mesma [120].

Tem como áreas fundamentais de aplicação a monitorização e controlo, *scheduling operation* de processos, preenchimento de formulários electrónicos, configuração e afinação de sistemas e em situações onde é dispendioso recolher informação [124].

O sistema começa por colocar os factos na memória de trabalho. De seguida, entra no ciclo de selecção onde procura, na memória de trabalho, os factos que satisfaçam as premissas das regras. Executa a regra seleccionada e coloca as suas conclusões na memória de trabalho [116].

O *backward chaining* ou orientado a objectivos, elabora a hipótese até aos factos que a suportam, e desse modo corrobora uma determinada conclusão [130]. Começa por converter a questão num objectivo/conclusão e com ele navegar por entre as regras até encontrar provas de suporte da sua veracidade.

Utilizado em diagnóstico de problemas, detecção de falhas mecânicas ou eléctricas, selecção de produtos e fornecimento de opinião, quando existe grande quantidade de informação e só algumas características particulares do sistema são de interesse [124] e em aplicações onde apenas é necessário provar a proposição em questão.

O Sistema coloca o objectivo na memória de trabalho. Procura regras com conclusão idêntica e coloca as premissas na memória de trabalho. As novas premissas inseridas passam agora a ser os novos objectivos, através de um processo de conversão, e toda a sequência é repetida até se chegar a factos verdadeiros.

"Estratégia mista" – em alguns sistemas é utilizada uma combinação dos métodos anteriormente introduzidos. Dada alguma suposição inicial, infere-se uma conclusão por encadeamento directo. Posteriormente, é aplicado o método inverso na procura de outros dados que confirmem essa conclusão. Em alternativa, pode-se começar com um objectivo, obtido por transformação, realizar encadeamento inverso até se obter alguma razão plausível e usar o mecanismo directo de modo a explorar as consequências destes novos dados. O segundo processo é também denominado de encadeamento oportunista - *opportunistic chaining* visto utilizar os dados à medida que ficam disponíveis.

3.1.9.6.1 Correspondência de Padrões - Pattern Matching

É um dos algoritmos mais conhecidos, tendo sido alvo de diversos melhoramentos. Encarrega-se de decidir quais das regras presentes na base de conhecimento se aplicam com o conteúdo corrente da memória de trabalho. Porém, na sua versão original é ineficiente uma vez que não só apresenta problemas de escalonamento como após selecção da regra a executar procede ao descarte da restante lista de regras habilitadas [118]. Como o tempo e a natureza cíclica do modo de operação limitam o número de factos modificados entre cada iteração a um valor diminuto, uma solução para os problemas anteriores surge com o armazenamento das avaliações precedentes.

O algoritmo RETE [131-133] em conjunto com mecanismo directo soluciona as limitações inicialmente enunciadas. A sua velocidade é obtida ao guardar informação das regras numa rede utilizada posteriormente na avaliação e propagação de alterações provocadas por novos factos.

Em termos sucintos o algoritmo constrói uma rede de nós interconectados – RETE network – representativos das partes condicionais das diversas regras. Os nós padrão, filtram os factos por categorias; os de junção juntam o resultado de vários nós padrão e executam as condições. Os nós terminais representam regras individuais.

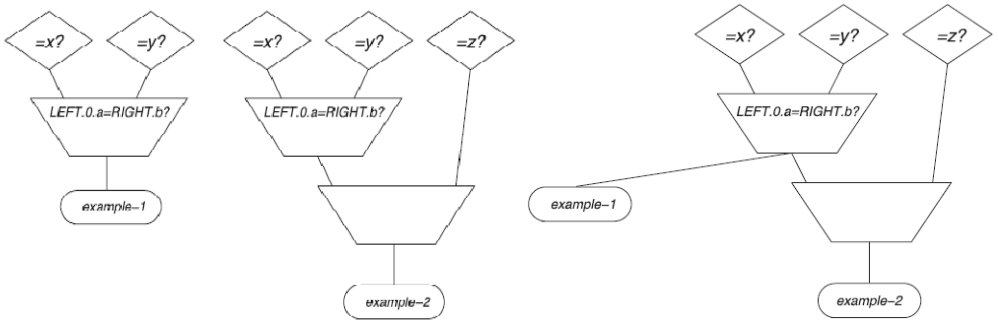

Figura 3.3 - a) Rede Rete não optimizada onde os diferentes nós padrão e junção se encontram separados o que origina algum processamento adicional. b) Versão optimizada com nós padrão e junção agrupados. Nesta modalidade os factos só são filtrados uma vez e os nós de junção identicos entre regras são reutilizados diminuindo as reavaliações [121].

À medida que novos factos vão sendo adicionados, são filtrados pelos nós padrão, e posteriormente enviados para os nós junção. Aqui, são guardados e testados perante as condições das respectivas regras. Caso se verifiquem , o resultado do nó de junção é dado pelo conjunto de factos acima deste. O processo continua até estar percorrido todo o caminho relativo à parte condicional ou até terem sido utilizados todos os factos.

3.1.9.6.2 Agenda

Contém regras, organizadas por prioridades, satisfeitas por factos da memória de trabalho com intuito de facilitar a escolha da próxima a ser executada. Em situações onde o número de regras habilitado a disparar é superior a um, esta entidade dispõe de mecanismos solucionadores de conflito responsáveis por decidir qual tem maior prioridade em termos de execução. O número e tipo de factores tomados em consideração no processo anterior são diversificados e divergentes entre sistemas. Alguns exemplos são: data de introdução, especificidade, ordem de dados, tamanho ou complexidade [121].

3.1.9.7 Editor da Base de Conhecimento

Destina-se a fornecer um meio conveniente, eficiente e automático de o utilizador introduzir e alterar todos os elementos constituintes da base de conhecimento, sem ser necessário recorrer a engenheiros de sistema [116, 124, 126].

Consoante a dimensão do sistema requerido e o nível de qualificação pretendido o conhecimento pode provir de fontes literárias, base de dados, registos históricos, casos de estudo e entrevistas com peritos. É no entanto necessário ter em consideração as limitações de cada fonte e a ocorrência de inconsistências.

3.1.10 Funcionamento

Apesar de o funcionamento de determinados elementos já ter sido introduzido, expõe-se agora todo esse processo de forma organizada.

As máquinas de regras têm um funcionamento cíclico, mas discreto e, apesar de existirem algumas variações principalmente a nível da avaliação e resolução de conflitos, o mecanismo básico constitui-se nas fases:

Acto de reconhecimento – Consoante o algoritmo utilizado o sistema procede à conjugação de um determinado número de regras com a memória de trabalho e perante as condições satisfeitas decide quais devem ser activadas. A lista resultante, denominada de *conflict set,* pode resultar exclusivamente da análise actual ou ser combinada com a de ciclos anteriores;

Selecção de regra – O *conflict set* é organizado de modo a dar origem à agenda, constituída pela lista de regras cujo RHS[17] irá ser executado. O processo, designado por resolução de conflitos, tem como objectivo seleccionar a regra prioritária, com recurso aos métodos anteriormente descritos.

Situação Acção e Resposta - Completam o ciclo, ao executar de forma sequencial as acções da regra de onde podem resultar alterações à memória de trabalho. Com a sua conclusão, a regra é removida da agenda e todo o processo é repetido de novo.

A Figura 3.4 auxilia na compreensão do processo antes descrito.

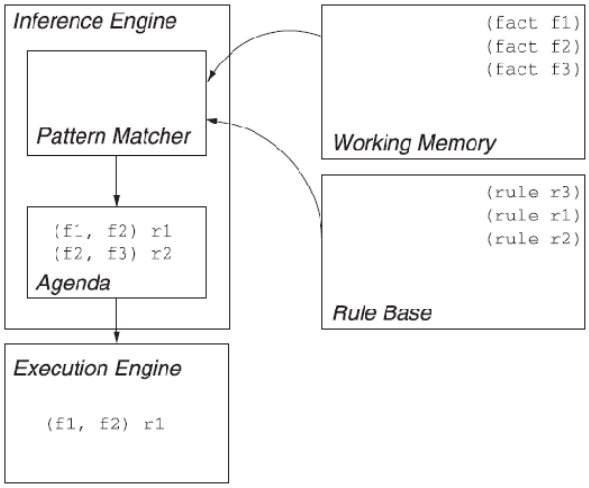

Figura 3.4 – Esquema simplificado de um sistema baseado em regras [121].

3.1.11 Áreas de Aplicação e Aplicações Correntes

Os RBS, só conseguem expor de forma plena as suas capacidades quando aplicados num domínio definido de conhecimento, a tarefa a realizar é clara, e é possível representar o conhecimento de forma *standard*, nomeadamente com recurso a factos e regras.

Em termos gerais, existe uma pluralidade de áreas onde estes mecanismos são utilizados, que vão desde a resolução de problemas de engenharia passando pela manufactura, telecomunicações, energia e transportes até ao governo, medicina, ambiente, agricultura, educação e ciência.

Em termos concretos, estes sistemas podem ser observados em aplicações como:

Selecção de componentes, equipamentos e *software* – Identificar de entre as possibilidades qual o produto mais adequado às exigências do cliente – SYSLAG, Achilles, DECADE.
Planeamento e planificação – Construir uma sequência de acções e assegurar a disponibilidade dos recursos necessários, com intuito de atingir um determinado objectivo.

[17] Right Hand Sid; Consequências

Controlo e Monitorização de processos – Governar o comportamento e proceder à comparação entre os resultados esperados e obtidos - PICON.
Assegurar qualidade – Participar em tarefas, propor práticas e fazer com que se cumpram os requisitos.
Automação de processos – Queixas, aprovação de créditos, vendas, análise de sinais, simulação cognitiva, sistemas com capacidade de aprendizagem.
Monitorização, Diagnostico, Detecção e localização de falhas – Determinar a causa do mau funcionamento, sugerir tratamentos e indicar medidas preventivas.
Classificação – De clientes, produtos e serviços, e riscos.

Não obstante à informação apresentada, áreas de aplicação adicionais podem ser encontradas em [116, 118, 126].

3.1.12 Factores Preponderantes à Utilização destes Sistemas

O formato e sintaxe de definição das regras estão dependentes do RBS utilizado e das ferramentas disponíveis para implementar e manter as regras.

No entanto, na maioria das aplicações, estes sistemas não trabalham sozinhos. A decisão de proceder à utilização ou integração de um RBS tem de ter em conta *custos vs benefícios*. *Custos* ao nível da curva de aprendizagem, da manutenção do pessoal e do esforço necessário ao desenvolvimento da interface de interacção. *Benefícios* incluem a abstracção facultada pelas regras, o seu isolamento do resto da aplicação e a capacidade da máquina controlar a avaliação e execução das mesmas.

Segundo Rudolph e Mahmoud[134, 135] existem algumas questões destinadas a auxiliar na decisão de utilização de máquina de regras:

O problema pode ser resolvido por programação convencional?

Se sim, então um sistema pericial não é a escolha mais adequada. Caso todas as respostas necessárias já sejam conhecidas bastará escolher numa tabela a adequada. Os Sistemas deste tipo são destinados a situações onde não existe algoritmo eficiente como solução, em situações onde é impossível conhecer todas as respostas à partida ou onde não é viável ter todas as combinações representadas.

O domínio/área está bem limitado/definido?

Caso a resposta seja negativa deverá ser utilizada outra tecnologia. Quanto maior for a quantidade de informação e subdomínios mais complexo o sistema se torna. Soluções divergentes podem ser apresentadas por áreas distintas com esforço acrescido ao nível da coordenação.

Há necessidade de um sistema pericial?

Estes sistemas resolvem as situações onde se encontra disponível pouca informação especializada. Logo é necessário ter em consideração o número de peritos existentes e a sua vontade em recorrer a estes sistemas. No final, o sistema só será utilizado e mantido caso exista uma real necessidade das vantagens e capacidades oferecidas.

Existe pelo menos um perito disposto a colaborar?

Nem todos estão dispostos a partilhar e testar o seu conhecimento com um meio computacional, pondo em causa a elaboração do sistema. A situação contrária também pode ocorrer com subsequente produção de resultados contraditórios.

O perito consegue fazer passar o seu conhecimento para o engenheiro de conhecimento - *knowledge engineer*?

Apesar de haver colaboração por parte dos peritos os termos técnicos das áreas são difíceis de entender e compreender.

De entre as diversas condições apresentadas, as provas da importância e necessidade do sistema junto com a existência de peritos dispostos a colaborar na construção são elementos fulcrais na hora de se decidir pela utilização de um sistema pericial.

3.2 Sistema de Fluxogramas

3.2.1 Introdução

A distribuição de trabalho existe desde os primórdios, altura em que a entidade dominante delegava tarefas aos súbditos. Na idade média, a situação descrita poderia ser equiparada à dos escrivães, sob o comando do padre superior responsável por proceder à distribuição das tarefas de verificação ou cópia dos manuscritos consoante a avaliação efectuada aos monges copistas. Com a evolução, o conceito principal de supervisores a monitorizarem a *performance* e atribuírem o trabalho com base nas capacidades, experiência e treino a vários recursos manteve-se, mas quem os implementava agora possuía ferramentas de automatização.

Ao se entrar na revolução industrial todas as organizações procuravam codificar e optimizar os seus processos a fim de conseguir realizar previsões e elevar a qualidade enquanto reduziam os custos. Todavia, na altura, seria impensável conseguir determinar quais actividades necessitavam de ser realizadas desde a produção até à entrega [136].

Os computadores iniciam a sua participação activa com a entrada na era da informação. São utilizados na gestão do estado e de pontos críticos do processo, permitindo informar sobre a situação das actividades e entregar notificações sobre a operação a realizar [136]. Nos últimos anos tem-se

assistido ao desenvolvimento de ferramentas que não só executam o trabalho como também efectuam a sua gestão. Passou-se de uma automação parcial na qual a execução do trabalho era auxiliada por programas externos para uma situação onde o processo é gerido por um programa de computador encarregue de distribuir trabalho, em alguns casos executá-lo, passá-lo à entidade seguinte, e monitorizar o seu progresso. Grande parte das alternativas disponíveis baseia-se numa aplicação separada a operar externamente com outras. O método considerado, apesar de aumentar a consistência, traz problemas a nível de integração. Outras opções menos utilizadas são os sistemas internos, que apresentam outro nível de sofisticação, e os intrínsecos considerados por alguns como o futuro. As suas funções, ainda mais alargadas, irão incluir a modelação, suporte, controlo e armazenamento do trabalho que passa pela empresa [137].

Com o número de aplicações a aumentar cresce também a sua complexidade e o milhão de utilizadores distribuídos por múltiplas organizações é hoje uma realidade. A gestão de recursos é agora uma prioridade, tarefa complexa quando os mesmos se encontram descentralizados, o universo de escolha é grande, bem como, quando existem vários papéis aos quais um recurso é qualificado. Tal situação ganha mais importância quando nos referimos a sistemas completamente autónomos, pois neles é necessário garantir a disponibilidade do recurso. *Standards* como OMG [138]e WfMC [139] ajudam no aspecto anterior ao definirem modelos de gestão de recursos.

Com este tipo de sistemas o trabalho deixa de ser incorrectamente atribuído ou empatado e as entidades gestoras podem concentrar-se na avaliação de *performance* e optimização de processos, em vez da comum e simples atribuição. Por seu lado, a entrega e monitorização do trabalho é feita automaticamente e os procedimentos são executados exactamente como definido pelo gestor. É também escolhida a entidade mais apta a realizar a operação, o trabalho encontra-se organizado por prioridade e, por último, existe suporte ao processamento paralelo.

Apesar dos elementos introduzidos e da necessidade constante de reorganizar os processos, só nos últimos anos, se assistiu ao nascimento de arquitecturas que promovem ou permitem a alteração dos mesmos ao nível do utilizador. As interfaces de alto nível permitiram a fuga à programação e a resolução de problemas de integração quer com recurso ao XML, destinado a fornecer um formato comum de informação, quer pela utilização do paradigma *webservices,* que junto com a internet disponibilizam uma plataforma distribuída de troca de mensagens [136].

3.2.2 Caracterização destes Sistemas

Ao longo dos anos, analistas e autores efectuaram diversas caracterizações destes sistemas e apesar de terem caído em desuso são um bom método de compreender as diferenças entre eles. O tipo de fluxograma a usar depende do objectivo a atingir. Daí ser comum as empresas utilizarem mais do que um tipo e em mais do que uma aplicação.

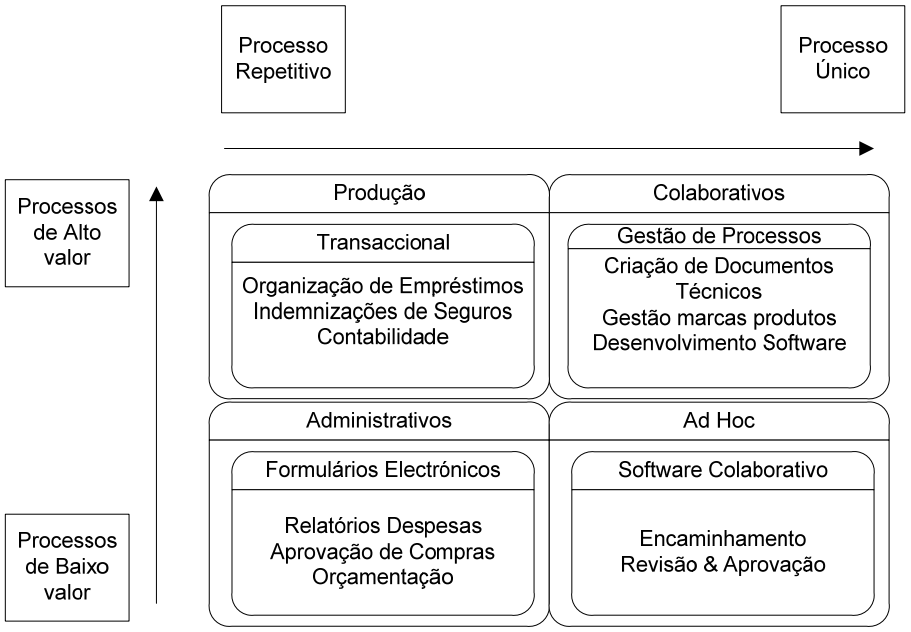

Figura 3.5 – Segmentação dos diversos tipos de fluxogramas consoante a utilização a que se destinam.

A separação mais habitual entre autores é feita em termos da necessidade de flexibilidade ou produtividade (Figura 3.5). Caso a primeira seja a pretendida, recorre-se ao fluxograma *Ad-hoc*, usado com frequência nas áreas profissionais e administrativas de uma organização. O tipo de fluxograma referido caracteriza-se pela existência de negociação e pela definição específica para cada situação. Normalmente, construídos sobre uma plataforma de e-mail, utilizada na distribuição de trabalho, oferecem bom controlo de processo e permitem criar e melhorar os processos de forma rápida, consegue-se assim dar resposta em tempo útil às situações que vão surgindo.

Associado ao anterior encontra-se por vezes, o Colaborativo de extrema importância numa empresa. Centra-se em grupos de equipas a trabalhar em conjunto para um objectivo comum. De igual modo as definições de processo não são rígidas e o volume de resultados não é o mais importante.

No extremo oposto, previamente definidos, e com canal de comunicação dedicado os fluxogramas de Produção suportam grandes volumes e como consequência conseguem reduzir os custos. Aqui, não existe negociação de quem realizará o trabalho ou de como ele será tratado, no entanto, podem ser adicionadas tarefas e procedimentos ao processo original à medida que vão sendo requeridas, por esse motivo possuem complexidade variável. Os objectivos de gerir grande número de tarefas similares e optimizar a produtividade são conseguidos com a automatização das mesmas em busca da automação total, onde o operador passa a intervir apenas em situações de excepção. Ao possibilitar a gestão de processos complexos e a integração em sistemas existentes a tendência é que seja embebido e funcione como máquina de regras.

O último elemento desta categorização é o Administrativo considerado como a junção entre Ad-Hoc e Produção. Neste, os passos estão pré-definidos mas é possível proceder à sua revisão ou alteração de forma dinâmica através da especificação de características dinâmicas. Como método de

entrega é utilizado o sistema de e-mail, quando o número de ocorrências é reduzido e um mecanismo proprietário em situações de elevada frequência. A sua característica mais marcante é a facilidade em definir processos, pois prescinde-se de alguma produtividade em prol de flexibilidade.

Diversas outras classificações estão disponíveis seja em termos da percentagem de automatização, do tipo de orientação utilizada i.e. humana vs sistema [99] ou documento vs. controlo vs. contracto [140], do nível de integração, da área de destino combinada com o meio comunicativo utilizado e com a orientação ao documento ou processo [141], do grau de distribuição [142], da granularidade da informação e da aplicação [143] e do tipo de aplicações/fins a que se destinam [144].

Contudo, com os processos a estenderem-se por organizações grande parte das alternativas revela-se inadequada. A nova realidade vale-se da duração do processo, dos limites organizacionais e das considerações em termos de funcionalidade como elementos de diferenciação. Na primeira, o tempo de vida médio de um processo determina o tipo de arquitectura necessária. Em aplicações como a montagem de componentes o processo é de curta duração, o trabalho a realizar é "empurrado" (entregue) e a coordenação e invocação de actividades provêm de máquina central. Já no extremo oposto com os Ciclos de Vida, é permitida a definição de processos longos, centrados em documentos ou num objecto, onde eles próprios se tornam o processo. Pelos limites organizacionais, efectua-se a distinção segundo a relação do processo com a organização. Por seu lado, a distinção ao nível das funcionalidades relaciona-se com as ferramentas destinadas a auxiliar a definição e execução de processos, nomeadamente as interfaces gráficas [145, 146].

3.2.3 Definição

Aquando da descrição pormenorizada dos sistemas de fluxogramas a sua compreensão só é possível se antes tiverem sido interiorizados alguns conceitos que, de forma comum lhe são associados. Pela entidade WfMC fluxograma, refere-se à versão automatizada de um processo, onde os documentos, informação e tarefas são transferidos entre participantes segundo um conjunto de regras de procedimento de forma a atingir ou contribuir para o objectivo pretendido [147, 148]. Surge, normalmente, associado à reengenharia de processos, onde existe o interesse por análises, estimativas, modelação, definição e a subsequente implementação do processo. Porém, pode também resultar da automação de um processo já existente.

O processo, resultado da associação entre a definição manual e do fluxograma encontra-se num formato que suporte manipulação automática, como seja a modelação ou promulgação por um sistema gestor. A sua definição, em linguagem gráfica ou formal, consiste numa rede de actividades interligadas, critérios e regras para controlo de progresso e informação sobre actividades individuais segundo aspectos como o participante e a aplicação [149].

O sistema de fluxogramas é o software responsável por definir, gerir e executar fluxogramas. A gestão é efectuada ao nível dos processos através do controlo do seu ciclo de vida, onde se inclui a

análise, desenho, operação, auditoria e melhoria. É também efectuada ao nível dos recursos, quer sejam humanos, máquinas, acesso a informação, consumíveis, aplicações ou outros processos. Sendo, ainda, efectuada ao nível do fluxo de processo, por aplicação de regras de procedimento destinadas a controlar a sequência e dependência entre actividades e recursos associados [143, 145, 147, 150].

A diversidade de opções disponíveis não tem neste caso, como consequência directa um grande número de aplicações incompatíveis entre si. Grande parte dos sistemas procura implementar funcionalidades comuns a fim de facilitar a interoperabilidade, possibilitar a integração e em casos extremos permitir a transferência de partes do processo.

A fim de resolver possíveis equívocos revela-se importante esclarecer a diferença entre *Workflow Management* e *Business Process Management* - BPM. Enquanto no primeiro o foco do sistema se encontra no processo, o BPM tem como elemento central a capacidade de gestão, destinada a provar a sua aplicabilidade na área empresarial e evitar os problemas inerentes à reengenharia. *Gestão de processos* engloba, então, a manutenção de versões, documentação, análise, controlo e modificação do processo em curso, com ferramentas capazes de ajustar os recursos e ajudar na distribuição de carga. Na vertente empresarial é acrescentada a capacidade de compreensão, monitorização e melhoria do ponto de vista comercial [145].

3.2.4 Razões de Existência

Este tipo de sistemas surgiu como resposta a um conjunto de necessidades não satisfeitas e particularidades não consideradas pelas aplicações existentes até à altura. De entre o grande número de factores podem destacar-se elementos como [146, 151]:

- Trabalho realizado pelo melhor participante – Sistemas deste tipo conseguem organizar, distribuir e atribuir o trabalho de acordo com algoritmos ou características pretendidas. Tal implica a existência de uma descrição para cada recurso na qual constam as suas capacidades, funcionalidade e particularidades.
- Conceito "Linha de montagem" – Em alternativa a ter pessoal qualificado, muito do trabalho pode ser processado de forma semelhante a uma linha de montagem onde cada fase é simples e especializada. O que se traduz numa maior flexibilidade e menor necessidade de treino.
- Sincronização – É dos passos mais dispendiosos pois, independentemente da altura em que chega um pedido, o mesmo, só pode ser executado quando estiverem presentes os elementos necessários à sua execução.
- Distribuição – Este tipo de sistemas é normalmente executado em ambientes complexos, logo a distribuição é uma obrigatoriedade nestes sistemas. Apesar da variedade de métodos disponíveis é vantajoso deixar a atribuição e a distribuição a cargo do sistema, segundo regras de optimização e mecanismos de resolução de conflitos.

- Completo – Cada sistema tem um método próprio de mover o trabalho entre as actividades constituintes do processo. Caso ocorram problemas, como a suspensão ou não finalização, grande parte dos sistemas permite adicionar tarefas à sequência inicial.
- Fácil acesso à informação – Com a execução do trabalho, diversos sistemas são consultados. Apesar da informação se encontrar distribuída o seu fácil acesso é garantido com as diversas interfaces disponíveis, compatíveis com sistemas antigos.
- Registos – Capacidade de armazenar o processamento realizado e introduzir comentários por parte do utilizador ou máquina. Elementos como a data, hora e entidade responsável irão facilitar a recuperação de erros.
- Recolha de trabalho em curso – Estas aplicações em adição a gerir o trabalho em curso também o identificam. Permitem assim, a sua localização e consulta de estado.
- Controlo Superior – Conseguido quer pelos procedimentos implementados quer pelos relatórios.
- Monitorização – A grande diversidade de informação recolhida possibilita a elaboração de relatórios, em questões de produtividade ou fiabilidade, destinados a realizar ajustes ou informar o utilizador de medidas a tomar.
- Evolutivo ou Escalonável – Possibilidade de adicionar ou refinar actividades ou sub-processos consoante as observações e aprendizagens realizadas.
- Flexibilidade - Novos requisitos, regulamentações ou tecnologias podem surgir ou ser impostos com impacto na definição de processo.

3.2.5 Vantagens

As aplicações baseadas em fluxogramas disponibilizam inúmeras vantagens em relação aos métodos de programação mais comuns. Inevitavelmente, algumas delas fundem-se com os requisitos introduzidos nos subcapítulos anteriores. Dispõem-se agora algumas consideradas de maior relevância.

- Menor necessidade de formação – O recurso a formalismos gráficos elimina a necessidade de o utilizador aprender linguagens de programação, o que torna o mecanismo de definição de processos mais fácil e intuitivo.
- Privacidade e segurança – Garante o acesso e alteração do processo e informação associada apenas a utilizadores devidamente credenciados. Deste modo, enquanto os quadros superiores dispõem de uma visão global do processo com posterior capacidade de visualização ou alteração de detalhes. Aos operadores comuns apenas lhes é permitido visualizar detalhes das actividades onde intervêm.

- Versatilidade e dinamismo – A simplicidade em definir e automatizar os processos e a possibilidade de proceder à sua redefinição durante a execução, facilitam a conformidade com requisitos presentes e futuros.
- Serviço de qualidade superior – As actividades e toda a informação por elas necessária é entregue ao recurso mais adequado em termos de qualificações.
- Maior rentabilização de recursos, produtividade, eficiência e qualidade – A automatização dos mecanismos de encaminhamento, reencaminhamento e processamento reduzem o tempo dispendido a valores essenciais.
- Impossibilidade de descarte – A rejeição ou atraso do trabalho deixa de ser considerada pois a entrega e gestão é realizada pelo sistema.
- Descentralização – Ocorre tanto a nível global como local pois o sistema encarrega-se de encaminhar o trabalho através da rede. Deste modo, os recursos necessários à execução não precisam estar todos associados à mesma máquina.
- Recuperação de erros – Entre outros, a informação guardada durante a execução permite que em tarefas complexas ou em caso de falha o processo possa ser restaurado no ponto onde tinha sido interrompido.
- Redução de custos – Os diversos benefícios apresentados antes garantem uma redução de custos.
- Independência - De forma semelhante à tecnologia anterior permite efectuar a separação entre lógica (fluxograma) e o suporte operacional (código) [147].

Como resultado de todos os benefícios em automatização, coordenação e colaboração este tipo de sistemas tem grande utilização na gestão de processos empresariais e industriais [152].

3.2.6 Desvantagens

Apesar das inúmeras habilidades e capacidades existem ainda questões por resolver. Algumas razões responsáveis pela resistência à adopção desta tecnologia são, por exemplo:

- Prescrição excessiva – Os programas utilizados sobrecarregam quem executa o trabalho com imposições e restrições. Com elas o participante fica impedido de realizar o trabalho da forma considerada mais intuitiva ou comum.
- Redução de autonomia – A participação num processo implica a conformidade com determinados parâmetros. Posteriormente existe o facto de as próprias actividades, quando entregues, possuírem directivas, procedimentos e *modos operandi* que devem ser seguidas durante a sua execução [146].
- Limitado / pouco adaptável – Se o caso apresentado não estiver de acordo com um formato esperado a aplicação pode ser incapaz de lidar com ele. Tal situação pode ter como causa

uma especificação vaga ou incompleta, um caso em que exista falta de informação ou o processo escolhido seja incorrecto para o trabalho.

- Teoria vs. Prática – Com a especificação comprovada em simulação não fica garantida a sua correcta execução numa situação real.

Contudo, as vantagens oferecidas por este tipo de sistema ultrapassam em larga escala as suas limitações, facto comprovado pela diversidade de aplicações existentes e pela sua utilização numa ampla gama de áreas.

3.2.7 Arquitectura & Elementos

Nos próximos subcapítulos serão descritos os componentes funcionais e as interfaces comuns constituintes deste género de sistemas. O seu entendimento é importante na compreensão dos termos futuramente utilizados, do funcionamento interno do sistema e da complexidade envolvida no processo de adicionar a capacidade de execução de fluxogramas aos denominados agentes de coligação.

A Figura 3.6 deixa antever um possível modelo geral. A concretização de cada um dos elementos pode ser efectuada em diversas tecnologias e com conformidade variável, relativamente à especificação global.

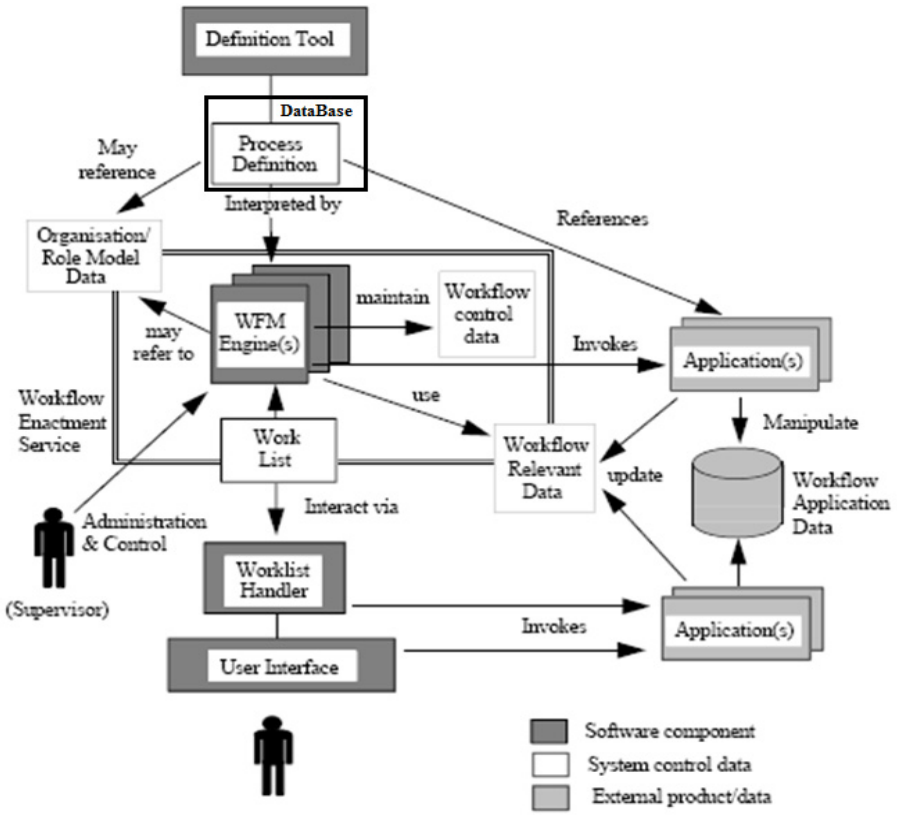

Figura 3.6 – Estrutura genérica de um gestor de fluxogramas [153]

Através da Figura 3.6 pode-se discernir 3 tipos de elementos. Os primeiros, com preenchimento a escuro, denominados por componentes de *software* fornecem suporte a várias funções no sistema de fluxogramas. As definições de sistema e informação de controlo, usada pelos elementos anteriores, apresentam-se no esquema sem preenchimento. Restam as aplicações invocadas e a base de dados, consideradas neste exemplo como parte do sistema principal.

As descrições e representação anterior permitem deduzir que um sistema deste tipo necessita de possuir, no seu essencial, uma ferramenta de definição de processos e diversas interfaces destinadas, entre outros, a permitir a recepção das definições de processos, a monitorização e supervisão por administradores, a atribuição de trabalho e a comunicação com outros sistemas semelhantes.

3.2.7.1 Definição de Processo

Um modelo, definido como uma representação abstracta do fenómeno real é formado pelo conjunto de características a representar. De modo semelhante um modelo de fluxograma realiza um processo de trabalho em grupo apenas com as características e actividades consideradas de relevância, escolha dependente do objectivo de construção. Tem como funcionalidades principais a descrição do processo ao grupo de membros e prever o comportamento da sua execução. A especificação pode, por exemplo, ser feita em termos de "funções", "participantes" ou "papéis", o que se traduz em representações com actividades associadas a entidades [154].

Uma linguagem de modelação tem como função concretizar a especificação do processo. De modo a ser aceite e ter utilidade necessita de ser flexível e extensível, dispor de mecanismos facilitadores do desenho e análise, e adaptação dos construtores ao domínio onde será aplicado. No entanto, as capacidades anteriores só são visíveis se os sistemas de promulgação estiverem aptos a tirar partido delas [155].

Segundo autores como Jablonski e Bussler um modelo de processo só é considerado completo quando contém a componente funcional, comportamental, organizacional, operacional e de informação, com função explicativa de aspectos sintácticos e semânticos, descritos exaustivamente em [156]. Um modelo de processo é, então na realidade, um subconjunto de sub-modelos interdependentes entre si [157].

Por seu lado os sistemas de modelação fornecem facilidades à concretização do processo, ao nível da distribuição de trabalho e da coordenação pretendida para pessoas e aplicações. Normalmente, na forma gráfica, armazenam as definições num formato destinado a facilitar a sua utilização por outras ferramentas, nomeadamente as de análise. Oferece, em adição às já descritas, a capacidade de interacção colaborativa e armazenamento de informação útil na elaboração de modelos.

Da utilização de aplicações externas na modelação e análise de processos, o WfMC reconhece a importância de separar o componente de desenho do de execução e publica uma linguagem para processos baseada em XML.

3.2.7.2 Serviço de Promulgação & Motor Fluxogramas

Promulgação ou *Enactment* refere-se à acção de codificar um procedimento num conjunto de directivas a serem executadas por combinações de humanos e computadores num sistema gestor[155]. Segundo o mesmo autor numa abordagem comum, *topdown,* a codificação do modelo é o último processamento a ocorrer antes da execução. Começa-se por proceder à sua elaboração, com intuito de automatizar um procedimento real. Só posteriormente à combinação e refinamento dos diversos diagramas é que se obtém a especificação a ser utilizada (3.2.7.1).

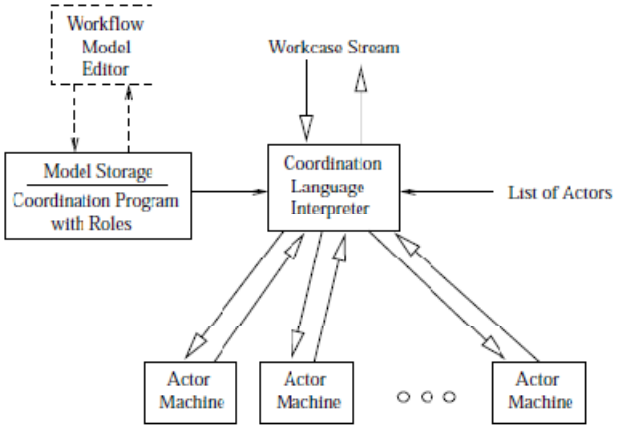

Figura 3.7 – Arquitectura, resumida, de sistemas de fluxogramas com foco na máquina de gestão [155]

O interpretador da descrição (vulgo executor), presente na Figura 3.7, é o elemento principal pois controla a criação de processos e gere a distribuição de trabalho tendo em consideração diversos parâmetros. Mantém, ainda, informação do estado de cada processo e actividade em execução, quando necessário invoca ferramentas de aplicação e possui métodos de recuperação de erros [147, 158].

Com a chegada de um caso, o executor determina qual dos passos atribuídos ao actor com o papel requerido deve ser executado primeiro. A escolha efectuada tem de ser capaz de cumprir a especificação. No final, o resultado é entregue ao interpretador/coordenador encarregue de indicar a próxima actividade. Em resumo, este componente associa actividades a "papéis" e identifica a unidade de computação (Actor) que mais tarde lhe será associada a fim de efectuar o processamento requerido na especificação da actividade.

3.2.7.2.1 Informação transitória e temporária

Informação disponibilizada pelos programas de aplicação, acessível à máquina de inferência. A definição do processo combinada com estes dados é utilizada no controlo de navegação entre as

actividades, bem como no fornecimento de informação relativo às condições de entrada e saída a elas associadas, e nas opções de execução das diversas tarefas ou aplicações constituintes.

3.2.7.2.2 Listas de trabalho

Em termos sucintos são simples listas de espera utilizadas por parte da máquina, na colocação de itens de trabalho nos diversos actores habilitados a executar as actividades necessárias. Quase sempre controlada pelo gestor de lista de trabalho pode, em alguns casos, ser acedida pelo utilizador para selecção do item pretendido [147].

3.2.7.3 Gestor de Lista de Trabalho

Componente de *software* responsável por orquestrar as interacções entre os participantes e o serviço de promulgação através da lista de trabalho. Pode ir desde a simples aplicação de aviso de trabalho em falta à sua repartição a fim de distribuir a carga [147].

3.2.7.4 Interfaces ao Sistema

Apesar de só agora serem referidas são conceitos de extrema importância sem o qual o mecanismo de promulgação não seria viável pois permitem o intercâmbio de informação com o mundo externo. Embora possa ser necessária alguma intervenção por parte dos programadores as interfaces disponibilizadas têm como função facilitar a realização de operações por qualquer utilizador credenciado (Figura 3.8). Independentemente desse aspecto, podem ser suprimidas ou combinadas com outras aplicações, como sejam as de elaboração de processo, a fim de estender o sistema original de forma quase imperceptível. Assim, enquanto por um lado se consegue uma melhor adaptação aos requisitos, por outro possibilita-se a integração de um componente adicional.

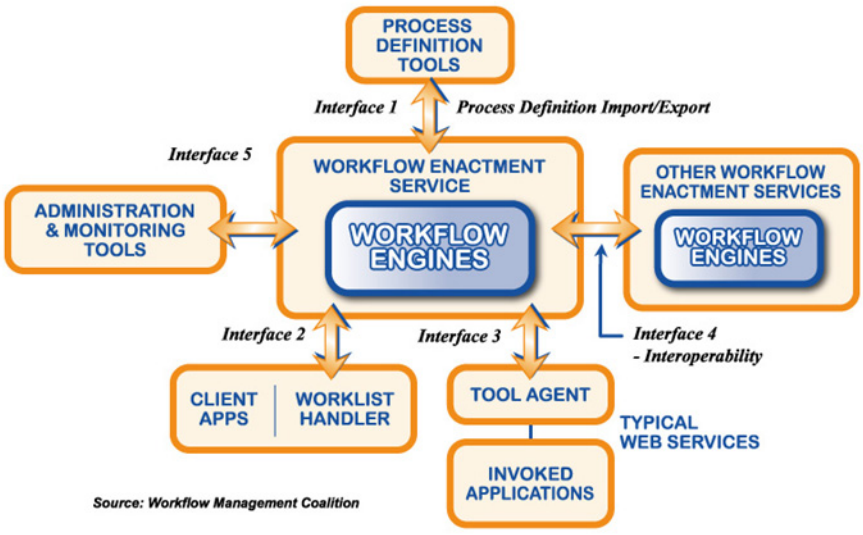

Figura 3.8 - Modelo de referência, onde se apresentam as diversas interfaces[153]

3.2.7.4.1 Primeira – Interface de processo

Tem como principal funcionalidade permitir o *loading* de processos definidos externamente ao sistema. Não obstante de ser o método mais comum, algumas implementações da especificação relativa a esta interface oferecem a possibilidade de representar os procedimentos e os recursos do processo. O objectivo principal desta interface é permitir o uso de diversas ferramentas externas de visualização e modificação de definições [150].

3.2.7.4.2 Segunda e Terceira - WAPI

Workflow Application Programming Interface combina a *interface* cliente e a de aplicações invocadas numa única, independente dos sistemas instalados. A implementação conjunta realiza a integração entre o sistema de fluxogramas e o "ambiente de trabalho". Do lado do executor fica habilitada a invocação de aplicações. Por seu lado por estas é permitido o acesso e fornecimento de informação. O nível de interacção pode ir desde o simples ver, seleccionar e completar até formulários e interfaces configuráveis pelo utilizador.

3.2.7.4.3 Quarta – Serviços externos de fluxogramas

Esta *interface* tem como propósito facilitar a interoperabilidade entre sistemas de fluxogramas, independentemente dos fabricantes ou da localização onde se encontram aplicados. Dependendo da conformidade com a especificação WfMC surgem diferentes níveis de interoperabilidade. Num dos extremos, marcados pela ausência de comunicação, encontramos os sistemas sem interoperabilidade. Por seu lado, no expoente mais elevado, encontramos aplicações diferentes a recorrer ao mesmo formato de representação dos processos, com a comunicação entre servidor e cliente de acordo com o *standard* e cuja interface apresentada ao utilizador é semelhante[150, 158].

3.2.7.4.4 Quinta – Interface de administração e monitorização

Como o próprio nome indica oferece diferentes níveis de monitorização e supervisão mas também permite actuar sobre os processos e elementos a ele associados. Deste modo, tanto é disponibilizado o acesso à história de cada caso como à informação global sobre o trabalho realizado. Para além disso é permitido terminar ou iniciar processos ou grupos de actividades manualmente e proceder a modificações em termos de papéis e actores. Por exemplo, ao nível individual, localiza e monitoriza as atribuições de trabalho, cujas regras de atribuição podem ser alteradas. Globalmente, é utilizado pelos integradores, na análise de produtividade e *performance* a fim de antecipar problemas e produzir alertas quando existem congestionamentos ou incumprimentos de prazos [150, 154, 159].

3.2.8 *Standards*

No começo da definição de *standards* por parte do WfMC a interoperabilidade era tida como o principal objectivo. Com a programação por objectos veio a constatação de que o sistema de promulgação deveria estar disponível através de interfaces que ocultassem a sua complexidade e, num esforço conjunto com a OMG, as mesmas começaram a ser definidas [158]. Na actualidade o WfMC dispõe de um modelo de referência que através das descrições das interfaces e conceitos introduzidos nos subcapítulos anteriores, nos permite ter uma visão global do sistema [147]. Por seu lado as especificações abstractas [160], a ele associadas, identificam as funções a implementar e os dados envolvidos. No nível mais baixo encontram-se os métodos destinados a como proceder à implementação das especificações segundo um conjunto de ferramentas, formatos e protocolos [161].

A utilização dos *standards* neste tipo de aplicações não se limita à implementação dos diversos elementos de sistema, e também está presente nas diferentes fases de descrição dos processos, expostos na Figura 3.9.

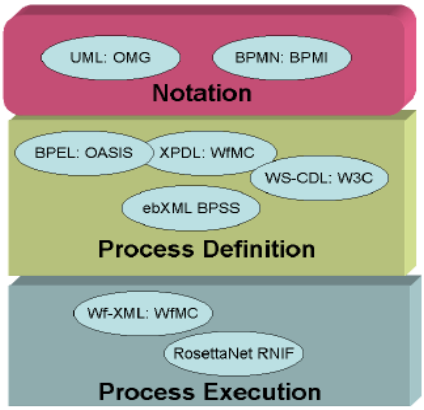

Figura 3.9 – *Standards* de representação dos processos nos diferentes níveis do sistema [157]

Na Tabela 3.1, classificam-se quinze alternativas de formatos destinados a proceder à descrição dos processos e permitir o seu intercâmbio entre aplicações, segundo treze pontos considerados de maior importância.

Tabela 3.1 - Formatos de definição/distribuição de processos [162]

	BPDM	BPEL4WS	BPML	BPMN	BPSS	EPML	OWL-S	PNML	UML Act.D.	WS-CDL	WSCI	WSCL	WSFL	XLANG	XPDL
Task I/O	+	+	+	+	+	-	+	-	+	+	+	+	+	+	+
Task Address	+	+	+	+	-	-	+	-	-	+	+	+	+	+	+
Quality Attributes	-	-	-	-	+	-	+	-	-	-	-	-	+	-	-
Protocol	+	+	-	+	-	-	+	-	-	+	+	+	+	+	-
Control Flow	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Data Handling	+	+	+	+	-	-	-	-	+	+	-	-	+	-	+
Instance Identity	+	+	+	-	-	-	-	-	-	-	+	-	+	+	-
Roles	+	+	+	+	+	-	+	-	+	+	+	-	+	+	+
Events	+	+	+	+	-	+	-	-	-	+	-	-	+	+	+
Exceptions	+	+	+	+	+	-	-	-	+	+	+	-	+	+	+
Transactions	+	+	+	+	+	-	-	-	-	+	+	-	-	+	-
Graphic Position	-	-	-	+	-	+	-	+	+	-	-	-	-	-	-
Statistical Data	+	-	-	-	-	-	-	-	-	-	-	-	-	-	+

Cada uma das linguagens relaciona-se com um determinado metamodelo através de operações de mapeamento. Efectua-se agora uma breve exposição do BPEL4WS e do XPDL. A primeira por ser considerada das mais completas, o XPDL devido a ter sido o formato utilizado no sistema ao qual se recorreu [104, 150, 162].

BPEL4WS – *Business Process Execution Language for WebServices* [163] -Também conhecido como BPEL é especificado apenas com base em esquemas XML, modela as tarefas como chamadas a *WebServices*, utiliza SOAP como protocolo de comunicação, o controlo de fluxo é orientado graficamente ou por bloco, a manutenção de informação é feita por variáveis, os papéis dos participantes são atribuídos por parcerias e possui métodos para lidar com eventos e falhas.

XPDL – XML *Process Definition Language* [148] resultante da evolução do WPDL[18], é o formato *standardizado* de definição e troca de processos proposto pelo WfMC. Caracterizada por ser facilmente compreensível e escalonável foca-se na descrição de requisitos e recursos. Inclui conceitos como endereçamento de tarefas; atribuição e recuperação de dados; controlo de fluxo, com transições do tipo AND e XOR; requisitos das tarefas; tempos de espera e de trabalho, a fim de definir os valores máximos associados às actividades; gestão e suporte de recursos; ferramentas de aplicação e informação; "papeis"; eventos e excepções. É também a única a introduzir estatísticas de duração e custo do processo.

A adesão, por parte dos sistemas implementados, a uma das alternativas apresentadas faculta uma maior valorização do investimento realizado, maior expansão e aceitação no mercado, bem como

[18] Workflow Process Definition Language

a diminuição do risco associado, um melhor suporte, uma maior facilidade de manutenção e um processo de integração mais simples.

3.2.9 Fases Necessárias à Implementação de um Processo

Como na maioria dos projectos ou tecnologias de informação que envolvem o controlo de outras entidades revela-se necessário efectuar um determinado conjunto de operações de forma organizada e estruturada de modo a conseguir obter resultados satisfatórios.

A sequência de passos necessários inicia-se com a clarificação da área de projecto onde se define qual o tipo de representação a utilizar e a gama de despesas previstas. Segue-se o Planeamento da implementação, de extrema importância pois quanto maior o cuidado em relação aos detalhes maior será a futura eficiência e utilidade do processo elaborado. A sua representação coerente implica a existência de uma compreensão profunda da situação real, conseguida à custa de observações, entrevistas, questionários, etc. Segundo Herbst[156], este é também um passo moroso motivado pela existência de diversos métodos e devido à confusão de termos ou à sua incorrecta aplicação. Ao dar-se por concluída a análise, são retiradas conclusões e sugeridas optimizações ao processo com vista a aumentar a produtividade. Os dados recolhidos juntamente com modelos UML serão utilizados na criação do processo.

O Desenvolvimento, também denominado de definição de processo, reúne e conjuga a informação obtida e esquematizada no passo anterior. Os fluxogramas são um método reconhecido na descrição das rotinas. O *standard* mais comum é dado pelo WfMC e especifica que as mesmas são constituídas por actividades automáticas ou manuais realizadas pelos participantes, algumas das quais podem encontrar-se agrupadas em sub-processos, para sequências de acções que ocorram diversas vezes [164].

Na fase de Teste o processo recentemente criado é sujeito a todas as eventualidades concebíveis. A ocorrência de situações não previstas pode ser um indicador da necessidade de ajustes ou alterações. A maturidade e o grande número de aplicações levam a que grande parte das aplicações já possua ferramentas de ajuda à construção e teste do processo.

A pré-implementação é também vital pois algumas fases do processo podem não ser automatizadas. Esta etapa dedica-se a fornecer formação, treino e aprendizagem às pessoas presumivelmente envolvidas. Consegue-se assim uma transição sem afectar, entre outros, a produtividade.

Depois de criada e testada, a definição do processo é colocada no WfMS responsável por proceder à sua instanciação e gestão, interacção com os participantes e invocação de ferramentas e aplicações de integração. A cada instância de processo irão corresponder diversas instâncias de

actividades. Estas, por sua vez, contêm itens de trabalho entregues aos participantes habilitados a concretizá-los.

Concluído o estabelecimento do processo e solucionados os problemas não detectados nas simulações e testes anteriores, a última etapa a realizar é denominada de pós implementação. Nela, procede-se à monitorização dos métodos recentemente implementados, a fim de recolher informações que permitam a obtenção de experiência e o aumento de produtividade.

A crescente automação na construção e execução dos processos traz consigo mais eficiência, porém, sozinha não fornece o máximo retorno. A reengenharia do processo, que procura optimizar a forma de trabalhar, em conjunto com o aspecto anteriormente referido, estão a modificar a forma como o trabalho é realizado.

3.2.10 Funcionamento

Este tipo de sistemas tem também um funcionamento cíclico, no entanto, a sua sequência de operações é totalmente diferente (Figura 3.10).

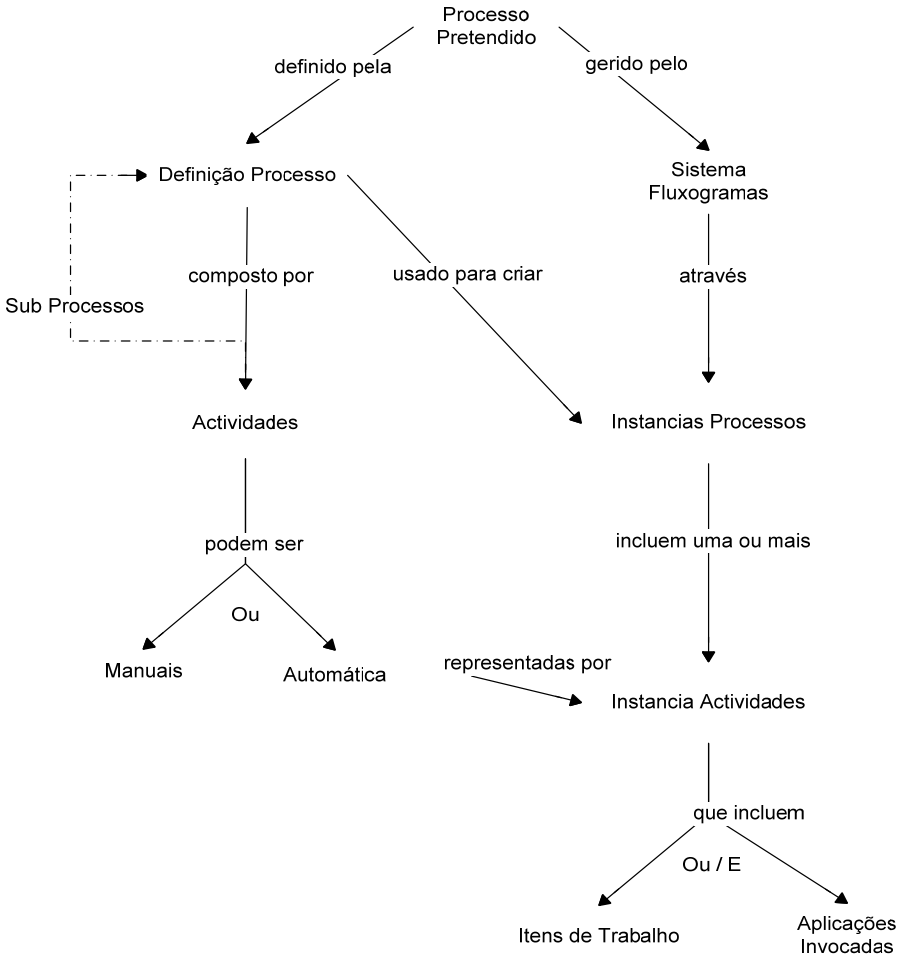

Figura 3.10 - Funcionamento global do sistema de fluxogramas

Dada como concluída a modelação do processo e colocada a sua especificação na base de dados a aplicação entra em modo de espera. A posterior necessidade de execução do mesmo é colmatada com a criação de instâncias. Na alternativa manual, essa situação ocorre normalmente pelo operador através de uma interface gráfica. Em oposição o modo automático, implementado no sistema a ser apresentado, após recepção do pedido despoleta uma procura e instanciação da definição adequada.

Com a execução da entidade a decorrer, a aplicação fica encarregue da sua gestão. Entre outros, determina qual a próxima actividade a executar, o participante à qual se destina e os dados requeridos. Após entregue, aceite e realizada, os resultados são devolvidos ao sistema dando continuidade ao processo [155]. Esta sequência de operações é repetida até a entidade gestora dar a instância por concluída.

3.2.11 Áreas de Aplicação

A utilização deste tipo de aplicações estende-se para lá do escritório, considerado por muitos a única aplicação, e engloba áreas tão diversas como vendas, contabilidade, engenharia, produção e fornecimento, prova de estarmos perante um sistema multi-funcional [136].

Algumas implementações concretas podem ser observadas no processamento de imagem, utilizado na análise e transferência dos dados obtidos. Também na gestão de documentos, nomeadamente no controlo do seu ciclo de vida, e em aplicações de grupo, destinadas a melhorar a comunicação entre membros. Ainda, em *software* de suporte ao projecto, destinado a encaminhar tarefas e informação entre entidades, e em ferramentas de desenho, associadas à reengenharia de processo. Os elementos apresentados mostram a multiplicidade de alternativas, mas também, a necessidade de aderir a *standards* de modo a simplificar o trabalho conjunto dos diversos elementos.

3.3 Sistemas Baseados em Agentes

Esclarecidos os conceitos principais associados a este tipo de sistemas e efectuada a sua contextualização, ficam por expor as características ou facilidades por ele disponibilizadas. Na actualidade, devido às suas características de abstracção, distribuição e interoperabilidade os sistemas baseados no conceito de multi-agentes são vistos como uma alternativa viável na elaboração de aplicações em ambientes distribuídos e heterogéneos [60], [57].

O grande problema com os ambientes multiagente correntes prende-se com o facto de nenhum dos *standards* existentes se encontrar suficientemente difundido/aceite e não existir nenhum ambiente adequado ao desenvolvimento de sistemas de agentes. Existem diversas entidades a trabalhar no aspecto de *standardização* tais como a OMG e a FIPA, e no desenvolvimento de ambientes de elaboração - DMARS, RETSINA, MOLE. De forma comum, com intuito de simplificar a construção

dos sistemas, as alternativas apresentadas baseiam-se em modelos de agentes, ferramentas e linguagens de comunicação predefinidas implementadas de forma distinta em cada sistema. A FIPA constatou a exiguidade destas considerações e como tal adiciona à sua especificação os agentes que considera necessários à gestão e manutenção do sistema. Em adição, mostra também a necessidade de uma ontologia partilhada destinada a permitir a interacção.

3.3.1 FIPA

Denominada de Federação para Agentes Físicos Inteligentes é uma organização que promove as tecnologias baseadas em agentes. As especificações FIPA representam uma colecção de *standards* destinados a promover a interoperabilidade de agentes heterogéneos e dos serviços por eles representados [67]. O objectivo da organização é através do uso de discurso cuidado e ontologias *standardizar* o uso de agentes [165].

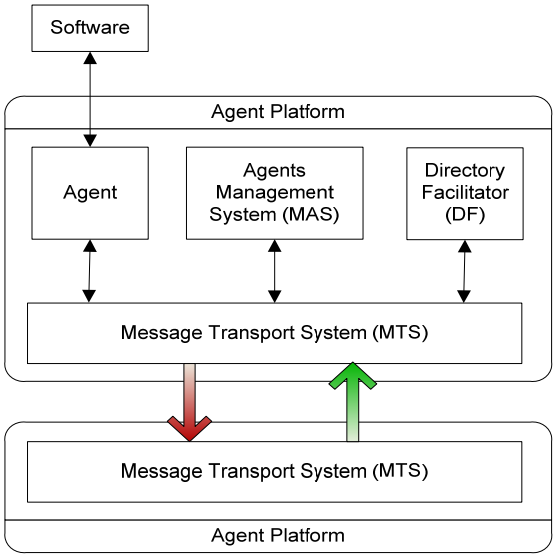

Figura 3.11 - Modelo de Referencia da plataforma de agentes FIPA

A sua contribuição ocorre principalmente ao nível conceptual e tecnológico, com a identificação dos diferentes sistemas passíveis de serem integrados. Logo existe uma preocupação em definir *standards* a um nível de abstracção elevado de modo a permitir alguma liberdade a quem desenvolve o *software*. Por esse motivo, estabelece um grupo de princípios e conceitos descritos nos capítulos seguintes e esquematizados na Figura 3.11.

3.3.1.1 AP – Agent Platform

Uma plataforma de agentes fornece uma infra-estrutura física onde os agentes possam ser executados. Nela, encontram-se englobadas as máquinas, sistema operativo, *software* de suporte aos

agentes, os próprios agentes e os componentes FIPA (DF[19], AMS[20], MTS[21]) destinados a gerir os mesmos. A predisposição interna dos elementos referidos não se encontra delimitada por nenhum *standard*, a sua concretização fica por isso a cargo dos programadores.

Os *standards* FIPA incidem apenas nos aspectos relativos à comunicação entre agentes de uma AP e para fora desta. A esse nível disponibilizam-se desde as AP´s de processamento singular até às completamente distribuídas. Estas últimas, úteis em situações onde é requerida a distribuição física dos agentes.

Pode-se então afirmar que os agentes FIPA existem numa AP e utilizam as suas facilidades para atingir os seus objectivos. Perante este facto, como entidade de *software* que são, o seu ciclo de vida tem de ser gerido pelo AP [166].

3.3.1.1.1 Agentes

Segundo a definição FIPA, um agente é um processo computacional com capacidades comunicativas autónomas, possíveis graças à linguagem de comunicação ACL. Visto como a entidade principal da AP, detém diversas aptidões cuja descrição se encontra armazenada no directório de facilidades (DF).

De forma a poder proceder à localização e disponibilização dos seus serviços, ou em último caso responsabilidade pelos actos praticados, cada agente tem de ser detentor de uma identidade única. No caso concreto dos agentes FIPA, esse objectivo é conseguido à custa do AID[22] que nomeia de forma inequívoca cada uma das entidades e possibilita assim a sua distinção dentro da comunidade.

3.3.1.1.2 AMS – Agent Management System

O AMS surge na forma unívoca mas a sua presença é imprescindível ao correcto funcionamento de uma AP. Este fica responsável por efectuar a gestão da plataforma através de operações como a criação, migração e término de agentes e detém autoridade para forçar as mesmas caso exista desrespeito pela entidade interveniente.

No seu interior mantém a lista de agentes pertencentes à AP. Essas descrições, conseguidas à custa dos AIDs das diferentes entidades, podem ser alteradas sempre que se relevar necessário. Todavia, essa modificação está dependente da aceitação do AMS. Com todos estes elementos, fica claro que o ciclo de vida de um agente termina com a remoção do registo do AMS.

[19] DF - Directório Facilidades
[20] AMS - Sistema Gestor de Agentes
[21] MTS – Serviço transporte mensagens
[22] AID – IDentificador de Agente

3.3.1.1.3 DF – Directory Facilitator

De forma usual, quando é feita referência a este componente da AP, estabelece-se a equivalência com um serviço de páginas amarelas, porém nesta situação a informação por ele disponibilizada destina-se a ser utilizada por agentes. Tem como principal função a manutenção da lista de agentes existentes na AP o mais próximo da realidade. Contrariamente ao AMS, podem coexistir diversos DF's na mesma AP. Existe inclusivé a possibilidade de se poderem registar entre si com o intuito de formar federações [166].

Quando um agente pretender publicar os seus serviços terá de proceder à localização de um DF e requisitar o registo da sua descrição. Por seu lado, o DF não consegue controlar o ciclo de vida dos agentes nem garantir a validade da informação por eles fornecida. É portanto, necessária atenção redobrada neste processo. Contudo, e apesar de a descrição ter de ser fornecida com todos os parâmetros, existe a possibilidade desta poder conter campos adicionais não *standardizados*, todos eles passíveis de serem alterados em qualquer altura.

3.3.1.1.4 MTS – Message Transport Service

O serviço de transporte de mensagens é responsável por efectuar a troca de mensagens entre agentes independentemente da AP onde se encontram localizados. Todos os agentes FIPA têm acesso a pelo menos um MTS e apenas as mensagens cujo destino seja um agente podem ser enviadas através dele [166]. Este é o método padrão de comunicação responsável por efectuar o encaminhamento de mensagens de forma fiável, ordenada e precisa.

A especificação FIPA também introduz uma linguagem de comunicação de agentes (ACL) utilizada na troca das mensagens referidas. A FIPA ACL é uma linguagem que descreve a codificação e semântica utilizada, o que elimina a ocorrência de equívocos, contudo não são referidos os mecanismos necessários ao seu transporte. Nela expõem-se também os diferentes tipos de interacção suportados que vão desde o simples protocolo, destinado à realização de um pedido, ao protocolo de negociação contract-net.

3.3.1.1.5 *Software*

No contexto corrente, o *software* pode ser visto como um conjunto de funções externas tornadas disponíveis à comunidade, graças à entidade agente. Na actualidade, a maioria dos agentes obtém os serviços através de outras entidades. Independentemente de estas serem ou não representadas por agentes, continuará a existir no futuro muito *software* cujas funções não são disponibilizadas através do paradigma introduzido. De modo a dar provas da sua utilidade é, então, imperativa a existência de uma interacção e controlo real por parte do agente sobre entidades de *software* como bases de dados, programas e *browsers* [167].

É possível proceder à categorização dos sistemas de *software* através da descrição da sua essência e do método em como proceder ao aproveitamento das suas funcionalidades. A razão por trás

desta catalogação prende-se com isso mesmo, utilização e partilha de recursos. Com a informação anterior os agentes podem adquirir dinamicamente novas capacidades, disponibilizadas posteriormente ao resto dos elementos [167].

4 Arquitectura

No presente capítulo demonstra-se como três tecnologias aparentemente díspares são integradas a fim de atingir a simplicidade na elaboração de processos e a detecção de novas capacidades. Nele começa por ser exposta a arquitectura envolvente à nova coligação criada. De seguida, são expostas algumas topologias implementadas e as razões do seu abandono, elementos de certo modo auto-explicativos da arquitectura actual. A quando da descrição concluída, será esclarecido como o sistema está destinado a funcionar.

4.1 Arquitectura de Suporte

O sistema implementado pretende elevar o conceito de coligação ao próximo nível e como tal reutiliza a estrutura envolvente já existente. O aspecto anterior agregado ao facto de uma coligação não ter utilidade sem a existência de membros leva à necessidade de entendimento do meio e entidades que o rodeiam

O trabalho anteriormente efectuado baseia-se no paradigma de agentes e coligações descrito em [4] e segue a visão CoBasa [12]. O objectivo desse projecto consistia em agentificar uma célula de manufactura, presente no laboratório UNINOVA, com o intuito de obter maior agilidade e versatilidade. Nela, cada um dos elementos activos da linha é abstraído por um agente que disponibiliza funcionalidades. A sua resposta aos novos requisitos de manufactura é dada pela combinação das capacidades dos membros numa versão rudimentar de coligação, que após alguma programação permite a execução de tarefas mais complexas.

A Figura 4.1 apresenta a arquitectura do sistema MultiAgentes já existente, concordante com o *standard* FIPA. Em termos globais, pela sua observação, conseguimos distinguir os agentes requisitadores e disponibilizadores de serviços. Ao nível da requisição de serviços a arquitectura possui os *agentes palete*. Caracterizados pela sua mobilidade, detêm em memória todas as operações a realizar, organizadas segundo uma ordem previamente estabelecida. A execução dessa sequência de actividades ocorre pelos disponibilizadores de serviços e tem início quando a entidade física palete dá entrada no sistema e efectua o primeiro pedido presente nas suas definições. Cumprida essa tarefa, nova se seguirá, processo repetido até toda a sequência ser dada como concluída.

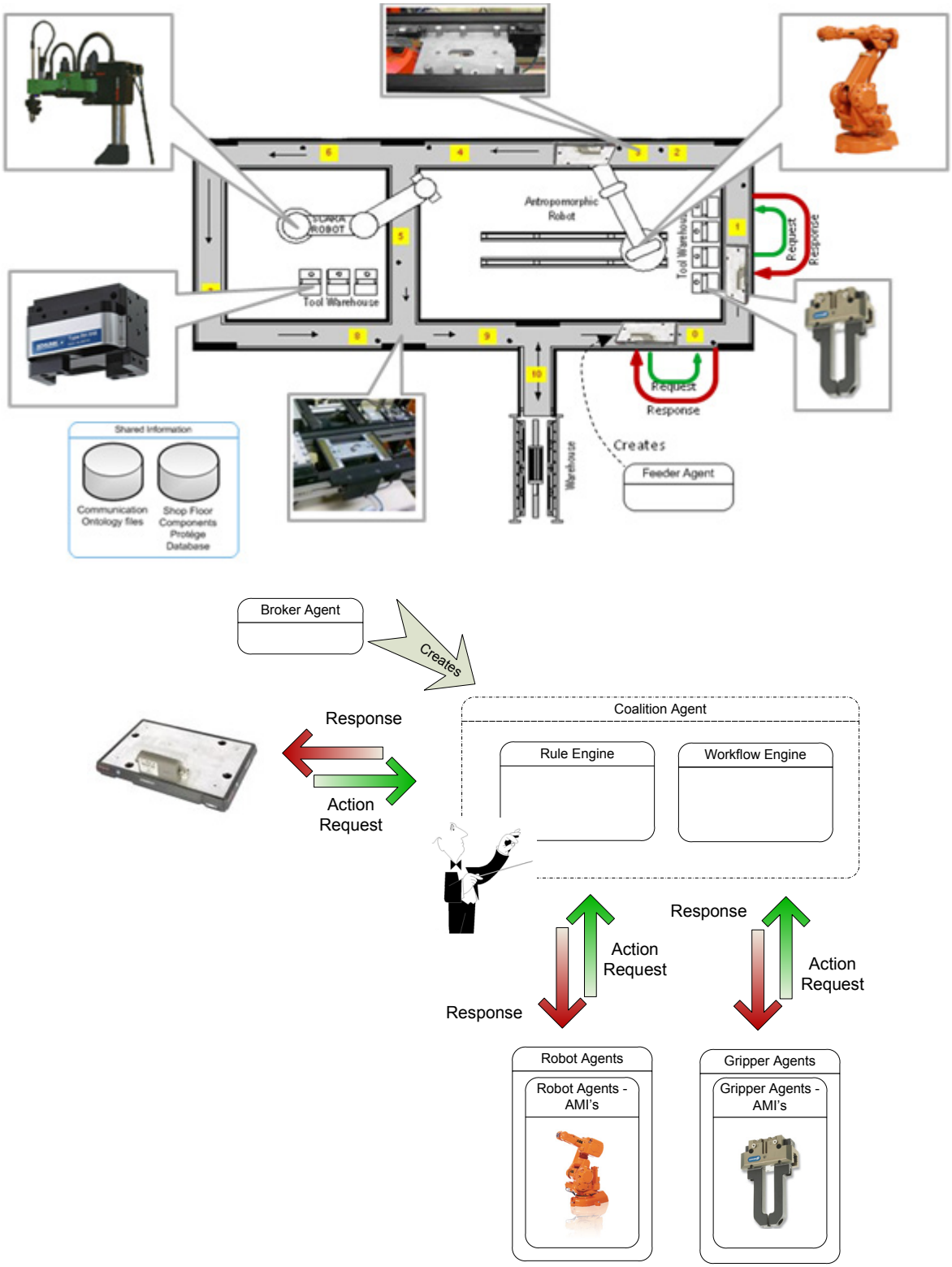

Figura 4.1 – (Acima) Arquitectura base da célula de manufactura experimental onde cada entidade física é representada por um agente. Nela é possível observar o agente palete responsável pelo pedido de operações, parte do caminho por ele percorrido e algumas interacções com os membros. Na posição 0, 1 e 2 o agente palete irá requisitar uma transferência ao *conveyor* actual. No entanto, na posição 3 o pedido já será direccionado para o agente coligação, apresentado mais abaixo na figura. Este agrupa e orquestra um conjunto de agentes com intuito de disponibilizar operações mais complexas.

Posteriormente, existem entidades como *robots*, *conveyors*, *transfers*, ferramentas e armazéns das mesmas encarregues de disponibilizar serviços. Estas, apesar de se encontrarem no estado activo, apenas desempenham as operações quando solicitado. Tome-se como exemplo o caso de uma palete localizada na extremidade de um *conveyor*. Perante uma consulta à sua sequência de operações fica a conhecer qual a próxima operação a efectuar. Caso a mesma implique a passagem da palete entre *conveyors*, é necessária a existência de um acto de comunicação informativo entre os dois intervenientes de modo a verificar a disponibilidade da entidade destino. Só após ter sido recepcionada essa confirmação ocorre o acto de transferência.

No entanto, na indústria a grande maioria destas entidades e capacidades por si disponibilizadas não tem grande interesse a título individual. Com o intuito de solucionar essa limitação surgem as coligações. Consideradas como um agrupamento de agentes, orquestram e coordenam as suas funcionalidades de modo a conseguir desempenhar tarefas mais complexas e úteis. Atente-se na operação de pegar e largar uma peça. Segundo esta visão a mesma é impossível de obter por uma entidade individual, pois na sua forma mais básica implica acções de uma ferramenta e de um robô. Nesta situação uma coligação irá proceder à selecção da entidade e actividade de forma concertada, com o intuito de conseguir obter o objectivo final de mover um componente entre dois pontos.

Existem ainda outros, os agentes *feeders*, *brokers* e AMI's que, apesar de não entrarem directamente nas categorias anteriores, desempenham um papel essencial no funcionamento da arquitectura. Os primeiros procuram facilitar a entrada de agentes na linha. São eles os responsáveis por instanciar na localização correcta cada uma das entidades *palete*, perante a sequência de acções por si armazenadas. A entidade *broker*, por seu lado, auxilia no processo de criação de coligações. Consegue-o ao disponibilizar numa interface gráfica, a listagem de agentes e coligações existentes através da qual o utilizador pode proceder à selecção das entidades desejadas e posterior criação de novas coligações. Por último, existem os *Agent Machine interface (AMI)*. Como o próprio nome indica, são agentes de interface, todavia na realidade, podem ser considerados agentes de baixo nível pois foram construídos e existem na arquitectura com o propósito de efectuarem a comunicação directa com o *hardware*. Para além de efectuarem uma abstracção directa das funcionalidades, comunicam com as entidades agentes genéricas para que estas possam disponibilizar as suas capacidades sem ter de se preocupar com a implementação propriamente dita.

4.2 Arquitectura Transitiva

Apesar das diversas arquitecturas e implementações realizadas, o propósito deste sistema sempre permaneceu idêntico. Fornecer um mecanismo simples, intuitivo, flexível e escalonável destinado à detecção de capacidades consoante novas entidades vão sendo introduzidas.

Posteriormente, deverá permitir a execução e definição, em tempo útil, da sequência que visa atingir essas novas competências.

Logo durante o início, a definição de uma arquitectura impôs responder a algumas questões de extrema importância, as quais iriam influenciar de forma clara a sua estrutura final. Qual das três tecnologias terá o papel central? Será o elemento dominante? Ao se considerar o meio envolvente deduz-se com facilidade, que a viabilidade apenas ocorre se o agente tomar o papel central como mais uma entidade disponibilizadora de serviços, numa rede onde os constituintes são exteriormente idênticos. Exposta a tecnologia responsável pela comunicação e entrega do trabalho aos agentes remotos, torna-se vital disponibilizar ao leitor informação da restante arquitectura interna.

A princípio apenas se verificou a necessidade de integrar uma aplicação no agente. A mesma deveria possibilitar a definição, carregamento e execução de processos, efectuar a gestão e manter um reportório dos mesmos. Idealizou-se, também, que esta camada de *software* devia permitir a inclusão no agente como biblioteca. A sua execução deveria ser despoletada pelo arranque do agente, no entanto, permaneceria em espera e apenas retornaria resultados quando nova informação fosse introduzida na sua base de conhecimento.

Com as primeiras implementações ficou demonstrada a inviabilidade da detecção de capacidades e definição de processos apenas com recurso a regras pois destruía um dos propósitos principais pretendidos para estes sistemas – simplicidade. Mais, com recurso a Rudolph[134], Mahmoud [135] e às proposições descritas em 3.1.12, esta seria uma das situações de não utilização. A obtenção de uma arquitectura praticável implicou a ocorrência de reformulações.

4.3 Arquitectura Proposta

Antes de se expor a arquitectura actual do agente de coligação criado, importa referir que as diversas mutações pelas quais passou não trouxeram implicações significativas ao resto dos agentes da comunidade. O sistema aqui implementado pretende dar continuidade ao trabalho descrito na publicação [4] com a introdução de novos mecanismos na execução de tarefas complexas. Em termos sucintos, a arquitectura envolvente é constituída por diversos agentes com capacidades próprias e normalmente representativos de uma entidade física. Cada um deles reporta as tarefas capazes de executar ao DF. Contudo, a concretização de actividades mais complexas requer algo ou alguém com capacidades de coordenação, é nesse ponto que entra o sistema criado - agente de coligação.

Tome-se como exemplo a operação de apanhar e colocar (*pickandplace)*, habitual em aplicações industriais. Facilmente se deduz que a actividade anterior resulta da combinação de duas operações já por si compostas, apanhar (*pick)* e colocar (*place)*. Estas sim, ao serem dissociadas dão origem a um conjunto de acções elementares tais como mover, mover linear, fechar garra, teste garra

fechada, etc., passíveis de serem executadas por agentes no ambiente. Nesta situação, o agente de coligação tem como função coordenar as operações disponíveis de modo a obter o objectivo referido inicialmente.

Na condição actual, os diversos agentes da rede ao serem iniciados começam por se registar a si e às suas capacidades no directório de facilidades (DF). Com eles, é também iniciado um *broker* destinado a facilitar a criação de coligações. Nessas entidades criadas *a posteriori* irá proceder-se à determinação dos processos passíveis de serem executados com os respectivos membros, seleccionados aquando da sua criação.

A primeira versão do agente de coligação data da mesma altura da restante rede. Construído apenas com propósito de provar o conceito de Agentes e coligações, peca pela rigidez, ausência de alternativas em caso de falha e pela extrema dificuldade em ser actualizada, pois toda a sequência de actividades encontrava-se em forma de código. A solução destes e outros problemas surge com a representação introduzida na Figura 4.2.

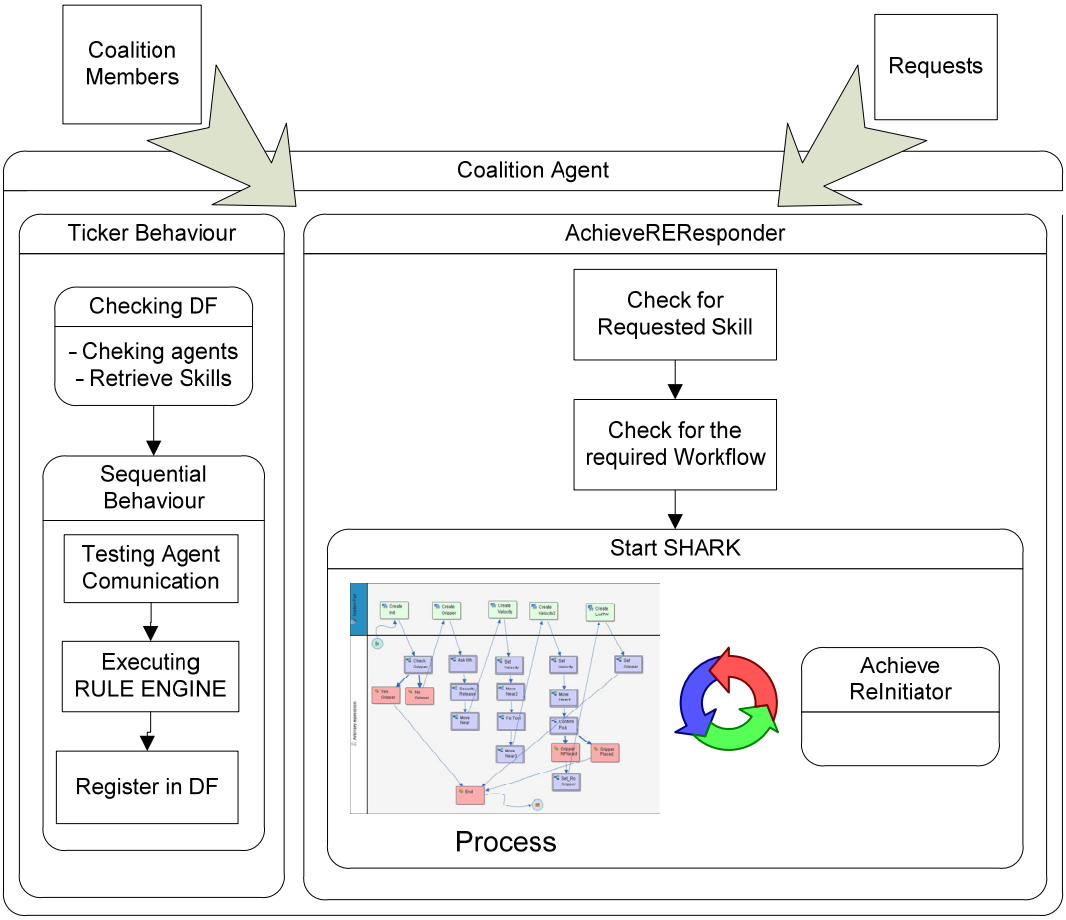

Figura 4.2 - Arquitectura de referência. Coalition Members – Lista de agentes representativa da coligação, entregue aquando da criação. Requests – Pedidos entregues à coligação durante o seu ciclo de vida.

Na nova arquitectura comprovamos o papel central do agente, incumbido de receber e responder a pedidos e delegar o trabalho que recebe dos níveis superiores. Porém, as semelhanças entre esta e qualquer outra implementação anterior cessam neste ponto, uma vez que apesar de se manter a utilização de regras e de a tecnologia adicionada posteriormente também ser acedida por bibliotecas, as duas repartem agora responsabilidades.

O sistema de regras passa agora apenas a calcular os processos possíveis de obter, quando antes também os executava. Faz isso com recurso a regras do tipo IF… THEN… ELSE. Esta facilidade disponibilizada trouxe consigo a flexibilidade, simplicidade e rapidez, essenciais na definição dos requisitos de tarefas complexas. O executor de fluxogramas, por seu lado, fica encarregue de distribuir as operações do processo a realizar.

As regras utilizadas são do tipo:

SE
Existir competência "mover"
& Existir competência "fechar garra"
& Existir competência "abrir garra"
ENTAO
Adicionar competência "pickandplace"

Aquando do arranque da coligação é também iniciado o sistema de regras cuja operação inicial consiste em carregar a lista de regras elaborada, sobre a qual é possível efectuar alterações e actualizações durante a execução. Novas ou antigas, todas as regras requerem factos e, como tal, o sistema possui um conjunto de métodos encarregues de os obter.

Concluída a fase de inicialização a computação das capacidades é efectuada em três fases, repetidas em intervalos de tempo regulares. Começa por consultar o DF sobre os agentes constituintes da coligação e caso algum já não se encontre registado é removido. Nos restantes é efectuado o teste de comunicação, constituído por um *PING,* ao qual os diversos agentes respondem com a sua disponibilidade actual. As habilidades fornecidas por cada uma dessas entidades resultantes dão entrada na base de conhecimento, como factos. O sistema de regras toma então o comando e consoante as regras introduzidas *a priori,* devolve as aptidões compostas passíveis de serem executadas pela coligação. Por último, proceder-se-á à sua recuperação e registo, no directório.

Suponha-se que a coligação se encontra em modo de espera e tem como membros ROBOT ABB e FERRAMENTA PGN1. Findado o intervalo temporal associado ao comportamento cíclico o sistema começa por consultar o DF sobre o ROBOT ABB e a Ferramenta PGN1. Em resposta, são fornecidas as aptidões de *move*, *movelinear*, *changevelocity*, *grip* e *ungrip* respectivamente. Com a lista armazenada, a secção seguinte envia uma mensagem, onde questiona a disponibilidade das

entidades. No caso onde ambos respondem com "activo&disponível" as suas aptidões, antes obtidas, são inseridas e processadas pelo sistema de regras que devolve *PickandPlace* como capacidade complexa susceptível de ser executada. Posteriormente, é feito o registo do processamento efectuado de modo a poder ser requisitado por outras entidades.

Noutra zona distinta, focada na comunicação com as diversas entidades remotas e parcialmente independente do sistema de regras, encontra-se integrado o executor de fluxogramas. É responsável por gerir as representações presentes na sua base de dados e interagir com o agente que lhe disponibiliza suporte, quer na indicação de tarefas a entregar às entidades remotas, quer por tratamento das respostas devolvidas. Consegue-o graças às interfaces construídas que aproveitam todos os mecanismos disponibilizados pelos agentes.

Contrariamente à tecnologia anterior e apesar de poderem surgir pedidos múltiplos, as execuções só se podem realizar quando existir confirmação de que estamos perante um pedido válido. Como tal, a arquitectura implementada faz uso dos resultados obtidos pelo sistema de regras e apura se a operação desejada se encontra na lista de tarefas. Em caso afirmativo, o bloco seguinte questiona o executor sobre se possui a descrição do processo pretendido. Só no caso de ambos os testes devolverem resultados favoráveis se pode instanciar o processo. Após instanciação ocorre a selecção e entrega da actividade ao executor. Posteriormente é recebida resposta, utilizada na decisão da próxima actividade a executar. Esta sequência é repetida até o processo ser dado como concluído ou até à ocorrência de erros.

Considere-se a situação na qual o sistema de regras computou, *a priori*, ser possível realizar *PickandPlace* e *ExchangeGripper* e uma palete, definida como agente requerente de serviços, se encontra prestes a solicitar a primeira aptidão. Com a arquitectura actual a coligação consulta os resultados disponibilizados pelo sistema de regras e verifica a disponibilidade da definição do processo *PickandPlace,* por parte do executor de fluxogramas. Na situação onde ambos obtêm resposta afirmativa e caso o processo não esteja já em execução, a palete é informada que o seu pedido foi aceite. É então criada uma instância da definição *PickandPlace*, semelhante à apresentada na Figura 4.3, onde se encontram as actividades a realizar, a quem entregar o trabalho, qual o caminho a seguir consoante as respostas recebidas e qual os parâmetros envolvidos em cada uma das situações. Neste caso concreto a primeira operação tem a denominação de *CheckGripper*, destina-se à entidade robô e tem como intuito confirmar que a ferramenta correcta se encontra acoplada. Tem como parâmetros o pedido a efectuar ao robô, o seu modelo genérico e uma variável onde irá ser colocada a resposta necessária ao posterior processamento.

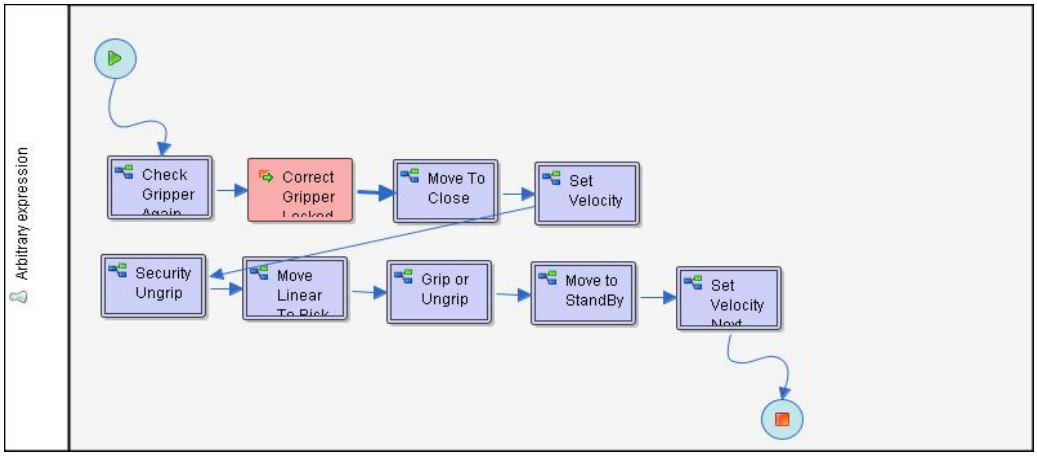

Figura 4.3- Definição do processo *PickandPlace*, onde se pode observar a sequência de actividades pretendida.

Depois de entregue à entidade remota é necessário esperar pela resposta de concordância ou não da execução e pelo sucesso da mesma. Cada actividade integra já o tratamento das diversas repostas passíveis de ser recebidas (Figura 4.4), em concordância com o protocolo FIPA. Não obstante, cada uma delas pode ser modificada e ou acrescentada conforme necessário. No final, e nesta situação, é recebido o "PGN1" na variável para o efeito. Esse valor é utilizado pela próxima actividade que decide qual a seguinte acção a tomar. A sequência repete-se até à conclusão da actividade "Set Velocity" altura em que o processo é dado como concluído.

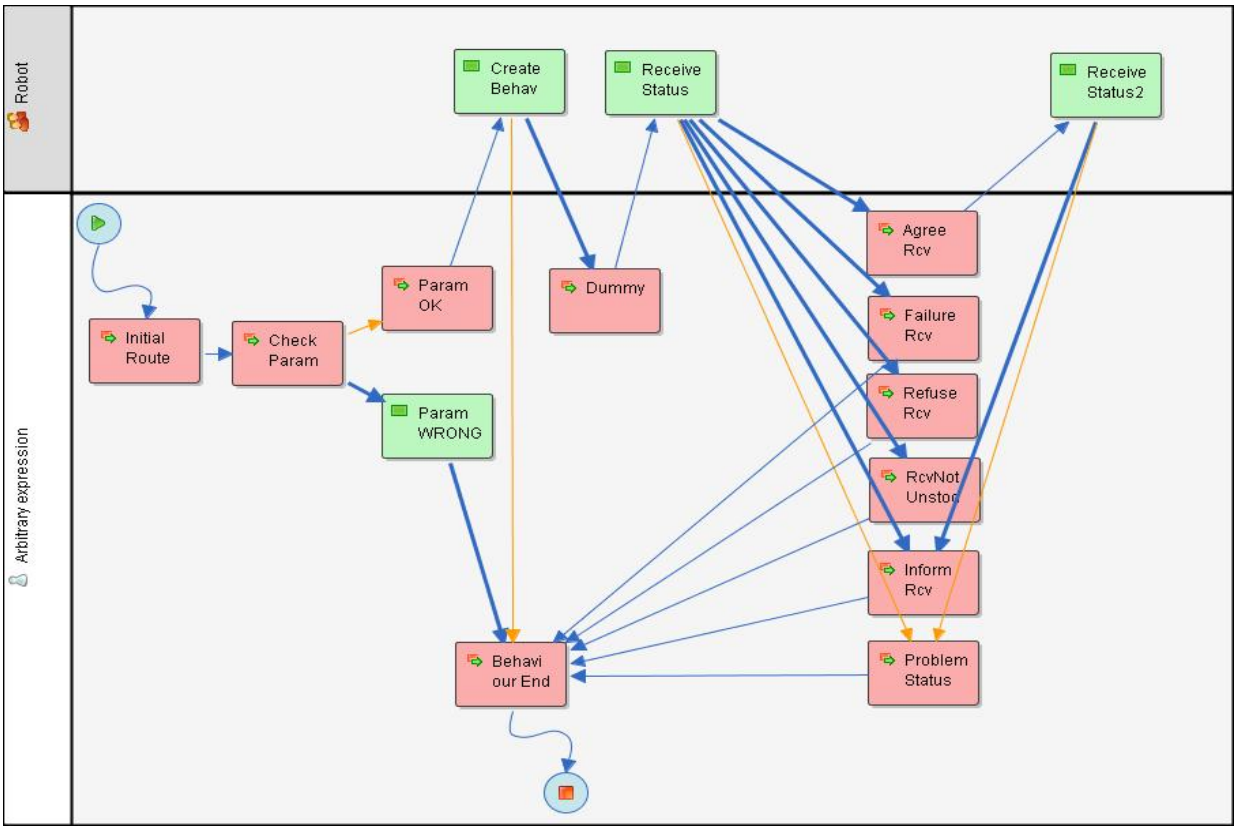

Figura 4.4 - Exemplo de implementação para uma actividade composta genérica

5 Implementação

Elucidam-se, no presente capítulo, os detalhes relativos à concretização do sistema. É feita referência não só às ferramentas utilizadas como às interfaces construídas em seu redor, destinadas a permitir a comunicação entre os diversos elementos. Antes porém é elaborado um percurso onde se expõe, em resumo, o caminho percorrido até à aplicação actual, os sistemas analisados e as razões porque foram abandonados em detrimento dos actualmente utilizados no sistema implementado.

5.1 Experiência

O Prolog começou por surgiu como boa aproximação à definição de processos. A sua velocidade, conforme comprovado por estudos e aplicações existentes[168], as múltiplas utilizações durante o processo educativo, factor abonador na velocidade de constituição e qualidade da implementação resultante, e a capacidade de simular o raciocínio humano indicavam ser a escolha adequada.

Enquanto decorria a primeira implementação, o trabalho sofreu uma bifurcação devido à pesquisa por uma aplicação que superasse o Prolog. Após algumas tentativas a implementação efectuada fazia grande uso de factos no armazenamento de informação. Entre outros, estavam abrangidas as coordenadas de agarrar - *grip* e as coordenadas de libertar - *ungrip*, a operação pretendida e as capacidades - *skills*, nas quais os parâmetros eram preenchidos quer pela ontologia quer pelo próprio agente. As regras por seu lado permaneciam inalteradas e impunham restrições à sequência de operações. Conseguia-se dessa forma obter a ordem de execução em tarefas compostas.

Revela-se agora, a título meramente ilustrativo, alguns excertos de código:

```
/*Factos          */
assert(pick (modelo,[lista tipo peças],[lista de posição])),
assert(pos (1,[lista posição])),
assert(pos(2,[lista posição])),
assert(gripper(modelo, [lista posição])),
…
/*Regras                  */
assert(PickPlace (grip, move, ungrip)),
assert(ChangeGrip (pick, move, place)),
assert(Follow(Pick, Move)),
…
```

Figura 5.1- Excerto de código relativo à implementação inicial realizada em InterProlog.

Tomadas as decisões relativas à linguagem, seria necessário um *software* que concretizasse a interface entre Prolog e Java. O Interprolog [169], obtido pela investigação paralela a decorrer, surge como uma alternativa promissora. Porém, durante a concretização, as poucas primitivas disponibilizadas e o seu baixo nível aliado à necessidade de elaborar uma interface intrincada, já disponível em algumas alternativas encontradas, levaram ao seu abandono.

Na nova aproximação é mantido o mecanismo anterior, contudo é utilizado o *Jboss Rules também conhecido como Drools* [170]. Uma aplicação mais desenvolvida e madura com compilador embutido que prima pelos métodos de elaboração das regras, desde o simples editor de texto ao programa proprietário. Os problemas expostos pela aplicação anterior são resolvidos com as interfaces de programação mais completas e pormenorizadas, como seja a que permite carregar um conjunto de regras definidas em tabelas Excel. A Figura 5.1 exibe um pequeno exemplo do tipo de codificação utilizado:

```
SE
        pos ACTUAL diferente pos destino 1º
        & existir skill "MOVE"
        & existir acção "position ready"
ENTAO
        o que se segue são "moves"

SE
        tipo peça, palete igual tipo peça, ferramenta
        & posição = robo
        & existir skill "GRIP"
        & existir acção "gripper ready"
ENTAO
        não é necessária operação e a gripper está pronta

When PickandPlace
        Then origem_perde & destino_ganha
When origem_perde
        Then procedimento requerido _grip_
When procedimento requerido _grip_
        Then check piece_ready & grip_ready & pos_ready
```

Figura 5.2 - Excerto de código exemplificativo do Drools

Com o decorrer da implementação constatou-se a inviabilidade da aplicação resultante se apenas houver recurso ao sistema baseado em regras. Segundo [134, 135] a utilização de regras deve, entre outras, estar associada ao conceito de mutação o que não acontecia na situação corrente, pois a ser utilizada na definição de processo as únicas alterações a ocorrer seriam ao nível das parametrizações. A aplicação dessa tecnologia era por isso incorrecta e uma nova implementação teve de ser realizada. Na versão mais actual é mantido o sistema baseado em regras no entanto, este funciona em paralelo com um sistema de fluxogramas.

5.2 Análise de Alternativas Disponíveis

5.2.1 Plataforma de Agentes - JADE

Existe uma multiplicidade quase infindável de sistemas de *software*. Caracterizados, entre outros, pelas suas interfaces e métodos de comunicação, todos possuem vantagens e desvantagens resultantes das escolhas efectuadas durante a sua concepção.

As tecnologias baseadas em agentes apesar de serem simples de entender e possíveis de implementar com as técnicas de programação comuns, só conseguem expor todo o seu potencial se dispuserem de uma plataforma adequada ao seu desenvolvimento.

O sistema construído será tão mais apto, quanto maior for o número de funcionalidades implementadas de entre os diversos requisitos impostos por estas plataformas. Por entre os diversos estudos efectuados por autores como Pavel [171], a plataforma mais consensual é denominada de Java Agent DEvelopment Framework [65]. A consideração anterior agregada ao facto da implementação que serviu de base à aplicação criada se encontrar desenvolvida na mesma plataforma foram motivos para a não existência de um estudo sobre as alternativas disponíveis.

O JADE é uma plataforma completamente implementada em java e cujo código é disponibilizado gratuitamente. Simplifica a criação de comunidades de agentes ao providenciar um conjunto de serviços e agentes, como o *naming service* (AMS), *directory facilitator* (DF), *message transport and parsing service* (MTS), bibliotecas de interacção e ferramentas gráficas de gestão e monitorização, concordantes com a especificação FIPA [172]. Ao se juntar à boa documentação e suporte, a facilidade de utilização e a baixa curva de aprendizagem obtêm-se um sistema com elevada aceitação e interoperabilidade. Uma análise detalhada desta *Framework* pode ser encontrada em [173].

A concorrência exigida em cada uma das entidades constituintes é conseguida com a introdução do conceito de "comportamentos" [174]. A sua utilização permite poupar recursos pois a mesma *thread* é utilizada nas diversas tarefas que repartem o tempo de execução entre si, consegue-se assim manter a execução de diversas acções em paralelo. Os actos comunicativos entre as diversas entidades ocorrem com a troca de mensagens segundo os pressupostos estabelecidos na teoria *speech-act*. FIPA-ACL é a linguagem utilizada na representação das mesmas. Relativamente ao conteúdo, o JADE segue a especificação do FIPA *semantic language* e fornece uma ontologia de gestão com possibilidade de integrar facilmente linguagens e ontologias proprietárias.

Como complemento às interfaces anteriores são também disponibilizadas ferramentas gráficas de diagnóstico *Dummy Sniffer* e *Introspector*. O primeiro permite a edição, composição e envio de mensagens com posterior recepção de respostas. Quando activo, regista as mensagens trocadas na

plataforma numa notação semelhante aos diagramas de sequência UML. O *introspector* permite controlar o ciclo de vida de um agente em execução, as mensagens trocadas e os comportamentos em execução.

5.2.2 Selecção de Sistema Baseado em regras

Uma vez que as versões iniciais não cumpriam os requisitos pretendidos de intuitividade, modularidade e capacidade de generalização, diversas alternativas foram analisadas na procura de uma solução adequada à nova realização em vista. Tarefa que se revelou de complexidade acima do previsto devido ao número de aplicações sempre crescente. Razão pela qual, também, não se optou pela construção de raiz deste sistema.

Dentro do grande número de características disponíveis pretendia-se, ao nível do sistema, a possibilidade de efectuar inferências sobre objectos e de as mesmas poderem ser realizadas directa ou inversamente. Na área de comunicação, as interfaces de programação deverão ser completas mas não de complexidade excessiva. De modo similar, as interfaces de interacção com o utilizador deverão ser intuitivas e possibilitar a definição de regras em diferentes linguagens e formatos. Em adição, o carregamento e modificação de regras sem ser necessário proceder a reinicializações também deverá ser contemplado.

Torna-se relevante exibir alguns exemplos e as razões porque não foram eleitos a fazer parte do sistema actual.

Mandarax + ORYX – API ao qual posteriormente foi adicionada a interface ORYX, que lhe permitiu suportar "linguagem natural". No entanto, apenas suporta inferências inversas, o seu funcionamento requer programas adicionais (tomcat, mysql), e é mais vocacionado a aplicações de rede com arquitectura de cliente - servidor, dai dar preferência a regras de derivação e possuir a máquina de inferência em separado. De referir também a preferência dada ao JDBC[23] em detrimento do JSR-94[24] [175].

Prova & TAKE – Evoluções do Mandarax, resolvem algumas das restrições indicadas antes mas por terem intuito diferente revelam-se inviáveis [176, 177].

Jeops – Projecto Português de pré-compilador apenas com suporte à inferência directa. Mais destinado a programadores utiliza um formato próprio na representação de regras [178].

[23] Java Data Base Connectivity
[24] Java Specification Request – Java Rule Engine API

JLisa - Framework baseada no LISA[25], um sistema de "regras de produção", considerado a extensão do Lisp e influenciado pelo CLIPS e JESS. A sua execução ocorre em java através do ABL[26]. Apesar de recorrer ao *Rete* e ser compatível com o JSR-94 utiliza um formato próprio na codificação de regras [179].

OpenRules – Mais destinado à área empresarial, detém como pontos fortes a utilização de ferramentas conhecidas e amplamente disponíveis e de recorrer ao Excel na construção de regras. Após alguns testes e perante regras complexas a habilidade descrita revelou ser mais um entrave do que uma vantagem [180].

Open Lexicon – Propõe-se como meio-termo entre os sistemas que utilizam linguagem natural, exigente ao nível de complexidade, e os que recorrem a linguagem técnica, apenas compreensível para programadores. Porém, a sua orientação aos *webservices*, o facto de não utilizar *Rete* e de as interfaces ocorrerem através de um browser tornam-no desadequado e excessivamente habilitado [181].

SweetRules – Segundo a própria definição são um "*conjunto de ferramentas integradas que procuram obter a semântica de regras e ontologias*". Consegue-o com recurso ao RuleML[27], SWRL[28] e OWL[29], este último *standard* de ontologias. As suas capacidades incluem a tradução e interoperabilidade entre diversas linguagens de regras e ontologias, suporte ao raciocínio directo e inverso e suporte a *webservices*. A necessidade de aplicações externas deixa antever a disponibilização de funcionalidades desnecessárias e complexidade acrescida, prova de se estar perante mais uma solução desadequada [182].

Zilonis – Tem como ponto forte a possibilidade de multitarefa, com ela fica eliminada a necessidade de replicar regras e a própria aplicação. A implementação recorre ao algoritmo *Rete* modificado e as regras são codificadas de forma semelhante ao CLIPS. Junto com a particularidade anterior a necessidade de utilizar um emulador de Linux, essencial na disponibilização de interfaces, restringe a sua aplicação aos programadores. Poderia, no entanto, com esforço extra ser utilizado [183].

[25] **L**isp-based **I**ntelligent **S**oftware **A**gents
[26] Armed Bear Lisp
[27] Rule Markup Language
[28] Semantic Web Rule Language
[29] Web Ontology Language

Algernon – Máquina de inferência que faz interface com Protege. Efectua a actualização de ontologias e bases de dados eficientemente e efectua processamento directo e inverso sobre as mesmas [184].

TyRuBa – Linguagem de programação lógica muito semelhante ao InterProlog. Logo pelos mesmos motivos não se coíbe com os objectivos pretendidos [185].

JTP – Sistema simulador de raciocínio, modular e orientado a objectos. Construído com o intuito de permitir uma maior e melhor orientação ao domínio, permite a adição de módulos de raciocínio à medida das necessidades. Infelizmente, é mais uma alternativa destinada a programadores e com funcionalidades desnecessárias [186].

JLog – Como sugerido pelo nome, constitui um interpretador Prolog codificado em Java. Em termos de funcionalidades encontra-se alguns passos à frente do InterProlog, no entanto o intuito é o mesmo [187].

Pellet OWL reasoner for java - Máquina de inferência sobre OWL[30]. Destina-se a aplicações que necessitem de realizar inferências sobre informação armazenada e estruturada em ontologias [188].

JShop2 – Aplicação que procede à decomposição de tarefas em subtarefas até chegar a primitivas directamente executáveis. Consegue-o por encadeamento de métodos onde se encontra descrito como as diversas operações se encontram relacionadas. Poderia ser uma alternativa possível, no entanto, a utilização do LISP[31] e uma interface elementar fazem antever complexidade acrescida na integração deste sistema [189].

Hammurapi – Similar ao *Jess* e *Drools* na utilização do *Rete*, da *interface* JSR-94[32], e no suporte aos dois tipos de inferência. Peca por estar vocacionado aos programadores, pois as regras são preferencialmente definidas em java. A sua constituição noutra linguagem apesar de referida não é desenvolvida [190].

Esper – Possui aplicação em situações onde existe grande fluxo de eventos a processar. Oferece filtragem, análise e resposta em tempo real aos mesmos mas, tal como noutros sistemas, não tem utilização no caso concreto [191].

[30] Web Ontology Language
[31] **L**isp-based **I**ntelligent **S**oftware **A**gents
[32] Java Specification Request – Java Rule Engine API

mProlog - Pretende disponibilizar uma máquina Prolog optimizada para a versão Compacta do Java - J2ME[33]. Apesar de construído com o objectivo de ser integrado numa plataforma 3-APL[34] (Lightweight Deliberative Agents), a sua interface genérica permite a integração em múltiplas aplicações [192].

OpenL Tablets – Sistema muito semelhante ao *Open Rules*, logo pelos mesmos motivos não aparenta ser uma solução adequada [193].

Abre rule – Embora não seja disponibilizado gratuitamente a semelhança nos objectivos, nomeadamente nos aspectos respeitantes à criação de agentes inteligentes por máquina de inferência, faz com que conquiste o direito a estar presente. Considerado como *Framework* é na realidade constituído por interfaces disponibilizadoras de serviços destinados, entre outros, à integração com agentes ou outros programas Java, à gestão de informação e ao controlo do mecanismo de dedução [194, 195].

Jboss Rules, Drools – Máquina de regras orientada à realização de inferências sobre objectos. Implementa uma forma modificada do algoritmo *Rete* adaptado à linguagem java. As transformações realizadas permitem a definição de regras de forma mais natural e a possibilidade de utilizar linguagem orientada ao domínio - DSL[35]. Foi construído de modo a ser extremamente fácil proceder à sua integração e devido aos ficheiros DSL tanto está direccionado para os programadores como utilizadores. Por fim permite a definição de regras em diferentes formatos e aplicações desde excel ou ficheiro de texto até à utilização de uma aplicação dedicada [170]. As características anteriores fizeram com que este seja um dos sistemas escolhido.

Por motivos óbvios, tanto nos sistemas baseados em regras como nos executores de fluxogramas apenas se expõem alternativas disponibilizadas gratuitamente. Não obstante, sistemas pagos foram também avaliados de modo a efectuar uma escolha adequada.

5.2.3 Selecção de Sistema de Fluxogramas

Contrariamente aos sistemas baseados em regras a escolha do executor de fluxogramas foi mais consensual. Para tal contribuíram factores como o número mais reduzido de aplicações existentes na altura e as funcionalidades por eles disponibilizadas.

Ao nível do sistema, as características procuradas baseiam-se em aspectos como o suporte à execução de múltiplas instâncias processuais em simultâneo e a possibilidade de poder alterar as suas

[33] Java 2 Micro Edition
[34] Artificial Autonomous Agents Programming Language
[35] Domain Specific Language

definições em qualquer altura sem ser necessário proceder ao término da execução. Relativamente aos aspectos de construção ou modificação das definições processuais, estes devem ser possíveis através de código ou blocos funcionais, onde se procede à definição das actividades constituintes e respectivos parâmetros. As alternativas eleitas devem ser de utilização intuitiva e permitir a verificação do trabalho realizado. Em termos de possibilidades comunicativas deverão ser disponibilizadas diversas interfaces com funcionalidades distintas. Permite-se assim a elaboração de aplicações capazes de efectuar a distinção entre operador e administrador, tanto a nível de grafismos como de operações permitidas.

Enunciam-se agora algumas das aplicações analisadas e os motivos pelos quais foram abandonadas.

BPEL – Especificação neutra desenvolvida pela OASIS[36] para especificação de processos segundo um conjunto de interacções entre *webservice* discretos. As suas diversas implementações defendem-na ao dizer que reduz os custos, a complexidade de integração e permite a construção de processos com diversos níveis de abstracção. A opção analisada BPEL-eclipse, permite a elaboração gráfica de processos e a análise de erros. Posteriormente, é também disponibilizada um interface muito completa. Na realidade o facto de facultar múltiplas funcionalidades tornam-no complexo e excessivo para a aplicação pretendida. Contribui, também, para esse desfecho o facto de ser principalmente orientado a *webservices*. No entanto, com modificações poderia ser visto como uma alternativa adequada [196, 197].

APACHE ODE - Executor de processos descritos no *standard* WS-BPEL[37]. Suporta manipulação de dados das definições, disponibiliza o seu descarregamento durante a execução, e dispõe de métodos de recuperação de erros e gestão de processos de longa duração. No entanto, aspectos como a necessidade de um máquina de *webservices*, destinada a permitir a sua execução, e a não existência de interface gráfica tornam-no inadequado [198].

Bonitasoft – *Software* que surgiu como evolução do projecto *Bonita*. Pretende ser uma solução igualmente poderosa comparativamente aos seus homólogos pagos, mas mais barata, flexível e fácil de integrar com a infra-estrutura já existente, nomeadamente bases de dados e sistemas de mensagens. Como tal, continua a ser disponibilizado de forma gratuita, no entanto, o suporte especializado já implica custos. As suas habilidades situam-se na definição de processos, através de uma interface gráfica avançada, e na possibilidade de importar definições em diversos formatos

[36] www.oasis-open.org
[37] Webservices BPEL

BPMN 2.0, XPDL e jBPM 3.2[38]. Por último, a simplicidade em carregar e iniciar os processos no executor torna esta aplicação numa das alternativas líder na área de BPM. As suas vantagens e semelhança ao *software* escolhido apresentam-no deste modo, como uma alternativa adequada [199].

ActiveBPEL – Sistema constituído apenas pela máquina executora de processos BPEL. Deste modo, a criação e alteração dos processos irá requerer a utilização de uma aplicação auxiliar. Independentemente destes aspectos, o facto de se recorrer ao BPEL impõe as mesmas limitações descritas anteriormente [200].

Imixs workflow – Mais uma solução resultante da evolução de um projecto de código aberto. Disponibiliza um gestor/executor de fluxogramas e um modelador de processos, os quais apesar de se encontrarem relacionados se destinam a funcionar de forma independente. Está também presente uma API muito completa destinada a simplificar o processo de integração do gestor em novas aplicações. No entanto, como mencionam os autores, é um projecto onde os fluxogramas são elaborados por humanos e para humanos, i.e. processos destinam-se a acções humanas e não a operações realizadas sem a sua intervenção [201].

Open business engine – Solução que apresenta algumas semelhanças relativamente ao sistema escolhido, *Enhydra*. Tal fica demonstrado pelo facto de ambos estarem em concordância com o *standard* WfMC. Existe também o suporte à execução síncrona ou assíncrona dos processos, a possibilidade de poderem ser constituídos por um conjunto de actividades manuais, automáticas ou uma combinação das duas anteriores, e de serem iniciados de forma manual ou por algum evento. À parte dos aspectos anteriores, o facto da instalação e configuração não ser intuitiva, da interface com a máquina ocorrer principalmente sobre a linha de comandos e de ser necessário recorrer ao editor Jawe para elaboração de processos, revelam ser imperativo a procura de outra alternativa [202].

jBPM – Gestor de processos mais destinado à área de negócios, no entanto facilmente pode ser utilizado na indústria. Procura distinguir-se entre a concorrência ao se focar tanto no utilizador como no técnico especializado. Tem como principais características o suporte para BPEL e jPDL[39], a disponibilização de uma API simples e a possibilidade de elaborar processos tanto numa aplicação separada como no eclipse. No entanto existe a necessidade de proceder a configurações e diversas adaptações nos aspectos relacionados com a linguagem (uma mais destinada a serviços e outra sem o grau de difusão esperado) [203].

Sarasvati – Solução com um núcleo simples que possibilita diversas implementações e a adição ou remoção de diversos módulos. Refere o suporte para *backtracking* e a existência de

[38] Java Business Process Management
[39] Java Process Definition Language

capacidades gráficas no entanto estas prendem-se apenas com a visualização de processos. Este último aspecto obriga à necessidade de proceder à elaboração das definições através de código XML [204].

com:cern – Executor de processos orientado a objectos com uma linguagem de modelação similar a um sistema de regras. Os aspectos anteriores traduzem-se na execução das actividades constituintes do processo apenas quando as suas pré-condições são correspondidas. Desse processamento resultam alterações no objecto que por sua vez geram pós condições. Actualmente a máquina executora encontra-se inserida num sistema modular com vista à adição de funcionalidades comunicativas, de modelação, administração/gestão, e criação de meta modelos [205].

Ruote - Solução para processos de longa duração que surge como evolução do projecto *OpenWFE*. Foca-se essencialmente na linguagem *Ruby*, pois segundo o autor, a sua intuitividade facilita a definição dos processos e das interfaces correspondentes, no entanto, também é possível recorrer ao XML. Face à existência de aplicações comerciais e o suporte para múltiplas instâncias do mesmo processo poderia ser uma aplicação adequada, contudo, a linguagem menos divulgada, a instalação por linha de comandos e o módulo gráfico limitado, obtido através de browser, não abonam a seu favor [206].

WfmOpen – Alternativa com origem na aplicação comercial WfMCore da empresa Danet GmbH[40]. Como se constata na página do projecto é uma implementação de máquina de fluxogramas baseada no J2EE, segundo as especificações do WfMC e OMG. Este componente, como lhe chamam, é constituído por um conjunto de interfaces que definem uma API destinada à gestão de fluxogramas. Apesar de se basear nas especificações originais, não se limita a estas e assim oferece uma maior escalabilidade e o suporte para SOAP. Apesar de aparentar ser uma solução adequada à integração a necessidade de proceder à instalação e configuração de componentes em separado levam ao abandono desta alternativa [207].

Taverna – Procura juntar um grande número de funcionalidades de modo a simplificar a descoberta, desenho, execução e partilha de fluxogramas complexos. Uma vez direccionado para a comunidade científica foi elaborado de forma a ser fácil e intuitiva a sua utilização, por esse motivo não exige grandes conhecimentos ao nível de computação. A solução disponibilizada contém aplicação cliente e servidor, no entanto, encontram-se ambas direccionadas para os *webservices*. Não obstante, dispõe de suporte, documentação e discussão adequada e possibilita a criação gráfica de processos com posterior verificação de erros nos serviços necessários [208].

[40] http://www.devoteam.de/

YAWL – *Yet Another Workflow Language*, como o próprio nome indica, constitui uma abordagem diferente a este tipo de sistemas através da utilização de uma linguagem para definição dos processos com a mesma designação. Apesar de menos divulgada, é já considerada uma alternativa viável ao BPEL e XPDL, pois oferece suporte nas diversas perspectivas de definição do processo (controlo de fluxo, dados, recursos), e permite a atribuição de tarefas a *webservices*, declarados em WSDL[41], a participantes humanos e a aplicações externas ou classes java.

A seu favor possui a facilidade de instalação, com recurso a ambiente gráfico, a possibilidade de converter processos de BPMN para YAWL e o suporte para simulação [209].

RunaWFE – Ambiente gestor de processos para o sistema gestor de processos jBPM[42] JBOSS. Da combinação do JBPM com alguns componentes adicionais resulta uma interface simples e intuitiva porem completa, sempre focada no utilizador final e disponibilizada através do navegador de internet. Em adição a oferecer suporte para *webservices* foi desenvolvida para facilitar tanto a instalação como a integração noutros sistemas. A referir também a existência de um elaborador de processos que recorre à linguagem jPDL[43] [210].

Enhydra Shark + JAWE – Recentemente renomeado para *Together Workflow Server* é uma máquina de fluxogramas modular, flexível e facilmente extensível e imbebível noutras aplicações. A sua implementação *standard* segue de forma completa a especificação WfMC e OMG, o que lhe confere APIs muito completas, usa o XPDL como linguagem nativa para a definição de processos e faculta "*tool agents*" para a execução de tarefas no lado do servidor. Disponibiliza também uma interface de administração e monitorização de extrema utilidade na verificação do estado dos diferentes processos a correr, e suporta comunicação com outros servidores. De referir ainda o facto de o núcleo deste executor ser utilizado como base noutras alternativas de sistemas apresentados.

Junto com esta ferramenta existe também o editor *Together Workflow Editor*. Totalmente de acordo com as especificações anteriores permite definir ou visualizar processos em modo gráfico ou através de código e proceder ao seu armazenamento de forma local ou remota. Em adição efectua a sua validação e possibilita a referência a outros pacotes e/ou definições [211].

As características anteriores e o facto de disponibilizar conjuntamente editor e executor levaram a que esta seja a aplicação escolhida para a execução de fluxogramas.

[41] Web Services Description Language
[42] Java Business Process Management
[43] Java Process Definition Language

5.3 Aspectos Práticos

Construídos os primeiros protótipos, resolvidos alguns problemas detectados e determinadas as áreas susceptíveis a melhoramentos, é chegada a altura de expor como a arquitectura actualmente válida foi concretizada. Com a implementação actual superam-se assim as limitações encontradas nas versões anteriores e introduz-se uma nova aplicação, destinada a gerir a execução de processos.

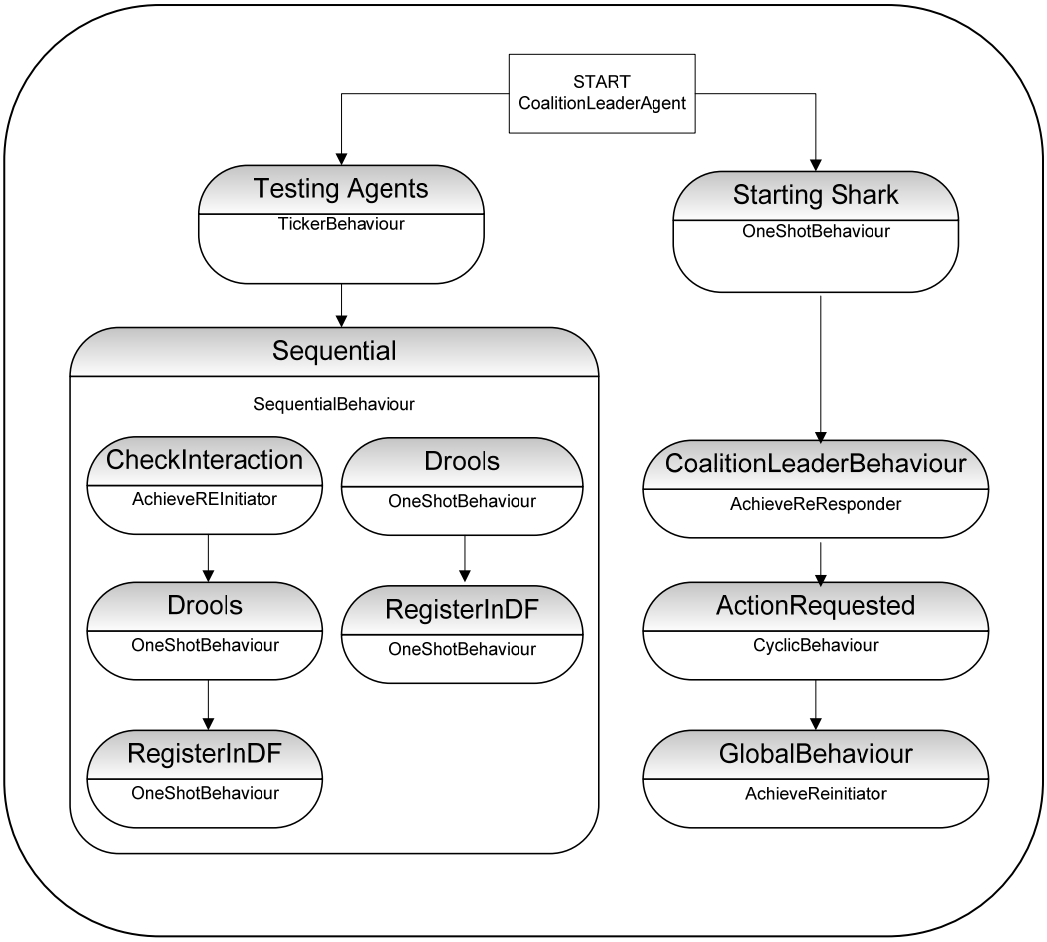

Figura 5.3 - Estrutura de Comportamentos Implementada

A Figura 5.3 expõe uma visão geral do agente implementado. Nele, expõem-se as aplicações introduzidas, todos os comportamentos utilizados e a forma como se encontram interligados. O facto de ambas as tecnologias não deterem pontos de contacto directos poderia levar à ideia de independência. No entanto, e apesar de até certo modo ser verdade, as informações disponibilizadas pelo sistema baseado em regras não teriam qualquer utilidade sem o executor de fluxogramas. Este último também não consegue realizar o trabalho sem os resultados do primeiro. Está então criada uma relação de interdependência.

5.3.1 Sistema Baseado em Regras

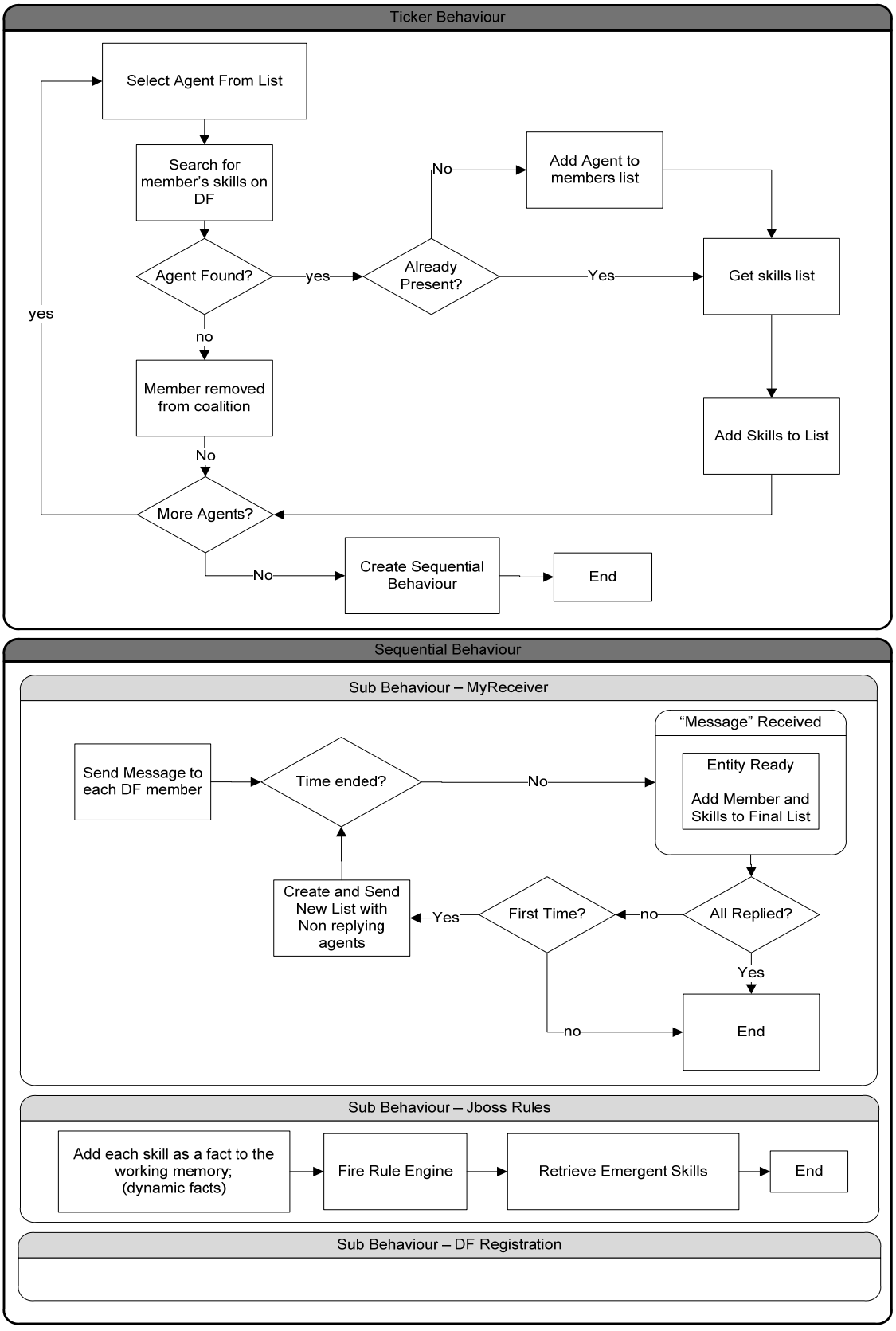

Figura 5.4 – Esquema de integração dos elementos referentes ao sistema baseado em regras no agente de coligação.

As responsabilidades actuais do sistema de regras sintetizam-se na computação de quais actividades o agente está apto a realizar em intervalos de tempo regulares. Tal como observado pela Figura 5.3 e Figura 5.4, essa operação implicou a sua integração com comportamentos do agente. O Jboss Rules facilita essa operação devido à presença de uma interface bem organizada e estruturada, ter como referencia o JSR94 mas, no entanto, não se limitar a ela, e ter exemplos úteis e explícitos em como uma integração deve ser realizada. A utilização de uma *thread* extra foi posta de parte não só devido à semelhança exibida com relação aos comportamentos, como também por interferir com o correcto funcionamento destes.

Como se pode observar pela Figura 5.4, o processamento começa no arranque do agente, com a adição de um comportamento que requisita a execução a cada 2 minutos (Figura 5.5).

```
public TestingAgents (Agent a, int check_time)
```

Figura 5.5 - Cabeçalho do comportamento TickerBehaviour cuja execução ocorre a cada "check_time".

O mesmo recebe a lista de agentes entregues pelo *broker*, representativos da coligação, acede a cada um dos elementos e questiona o DF sobre as funcionalidades por ele disponibilizadas. Caso alguma entidade já não se encontre disponível, é removida do grupo resultante.

Concluída a consulta apenas fica provado o registo dos agentes no DF e portanto a sua correcta inicialização. De modo a eliminar ou pelo menos reduzir a ocorrência de comunicações inválidas e falsos cálculos, foi adicionado o comportamento sequencial – s*equentialbehaviour* (Figura 5.6). Com ele aumenta-se de forma significativa a fiabilidade do processo comunicativo com as diversas entidades e logo a exequibilidade dos processos complexos determinados.

```
SequentialBehaviour seq = new SequentialBehaviour();

       if (myGui.getDroolsDef()){
              myGui.writeResume("TESTING REAL COMUNICATION WITH AGENTS");
              seq.addSubBehaviour(new CheckInteraction(this,alive_msg,members_DF));
              seq.addSubBehaviour(new Drools(this, Members_Com));
       }else{
              Actual_Members = members_DF;
              seq.addSubBehaviour(new Drools(this, members_DF));
       }
       seq.addSubBehaviour(new RegisterInDF(this));
       addBehaviour(seq);
```

Figura 5.6 - Excerto de código referente ao comportamento SequentialBehaviour. Consoante a opção seleccionada na interface gráfica poderá ou não ser efectuado o teste de comunicação com cada membro da coligação.

O excerto de código da Figura 5.6 expõe as duas alternativas de computação e registo das habilidades, distinguíveis pela existência ou não do teste de comunicação com os agentes.

Na situação onde o botão *Test Com* da Figura 5.9 se encontra activo a verificação é realizada com recurso ao *Reinitiator - CheckInteraction*, que recebe o template de mensagem a utilizar e a lista de membros à qual a enviar. Dentro do mesmo são inseridos os destinatários junto com o conteúdo da mensagem e um mecanismo iterativo disponibiliza a flexibilidade necessária.

O preenchimento do conteúdo é feito de acordo com a definição do conceito *CheckAction*, presente na ontologia (Figura 5.7). Os dois campos constituintes, *PresentStatus* e *Present* recebem respectivamente o valor de *status* e *present*.

```
CheckAction information = new CheckAction();
       information.setPresent(PRESENT);
       information.setPresentStatus(STATUS);
```

ComponentsVocabulary.java

```
public static final String PRESENT = "present";
public static final String STATUS = "status";
```

Figura 5.7 – Campos do conceito *CheckAction*, quadrado se cima. Template de preenchimento para os diversos campos da classe *CheckAction*, quadrado de baixo.

Por seu lado, os agentes remotos agora aptos a receber este tipo de acção, devido também à inclusão do conceito *CheckAction*, na resposta devolvem os campos anteriores com *check* e a sua disponibilidade actual - *busy, aborted, ready*.

```
Calendar expires = Calendar.getInstance();
expires.setTimeInMillis(expires.getTimeInMillis() + 1000);
request.setReplyByDate(expires.getTime());
```

```
protected void handleAllResponses(Vector responses){
```

Figura 5.8 – Excerto de código relativo à temporização utilizada com intuito de evitar que o agente fique indefenidamente à espera das respostas de todos os membros (quadrado de cima). O quadrado de baixo apresenta o cabeçalho do método responsável por tratar das respostas recebidas e caso necessário proceder ao reenvio.

No término do intervalo de tempo, o método da Figura 5.8 reenvia o pedido às entidades cuja resposta ainda não tenha sido recebida. As mesmas são removidas da lista caso a segunda análise as continue a dar como ausentes. Este mecanismo iterativo fornece uma hipótese adicional de os agentes provarem a sua existência e oferece alguma protecção contra problemas de comunicação momentâneos.

Com a lista de agentes actualizada, o sistema de regras é o próximo passo. A classe, de execução singular *Drools*, carrega o ficheiro de regras e os factos, materializados pelas habilidades de cada um dos membros remanescentes da coligação original. Durante o cálculo, os resultados obtidos são adicionados à memória de trabalho, com intuito de possibilitar a obtenção de novas competências que deles necessitem. De modo semelhante, é criada uma lista com todas as capacidades obtidas, útil no posterior processo de registo no DF.

Se se excluir a situação onde o ficheiro de regras é carregado durante a inicialização, a actualização da base de conhecimento apenas ocorre quando tal for requisitado pelo utilizador, por meio da interface gráfica. Caso seja esse o seu intuito, será necessário indicar a localização do novo ficheiro de regras e mostrar a intenção de actualizar a memória com o botão de *Update Rule***.** Essa operação cria um pacote com as novas regras elaboradas e procede à actualização da base de conhecimento (Figura 5.9).

```
source = new InputStreamReader("/rules_definition.drl") );
PackageBuilder builder = new PackageBuilder();
builder.addPackageFromDrl( source );
Package pkg = builder.getPackage();
RuleBase ruleBase = RuleBaseFactory.newRuleBase();
ruleBase.addPackage( pkg );
```

Figura 5.9 – Sequência de métodos executados quando se pretende carregar um novo ficheiro de regras (acima). Fragmento da interface gráfica, com os diversos botões e caixas de texto, associados ao sistema de regras (abaixo).

O mecanismo supra descrito apenas tem utilidade se houver forma de constituir regras. Uma das características que levou à eleição do Jboss Rules como o sistema de regras foi a possibilidade de definição das mesmas em diversos ambientes e com linguagem de domínio. Com ele, a descrição tanto pode ocorrer num simples ficheiro de texto como no ambiente de programação Eclipse. Em adição, existem os ficheiros tradutores destinados a permitir a criação de regras com termos adaptados ao meio.

A implementação mais comum ocorre num ficheiro DRL, com uma linguagem muito similar ao *Able* e logo ao próprio *JAVA* em si, inclusive no método de inclusão de classes necessárias ao correcto funcionamento das regras. Na definição das mesmas recorre-se ao conceito de capacidade/aptidão, todavia, optou-se pela utilização de uma classe genérica como alternativa à distinção entre capacidade simples e complexa, pois tal assunção é relativizada pelo grau de granularidade considerado. Denominada em inglês de *skill* possui dois campos, o nome e a lista de tarefas que a compõem. Os diferentes métodos por ela implementados oferecem às regras a possibilidade de aceder e modificar os objectos existentes.

```
rule "PICKANDPLACE"

       when
              exists skill(name == "Move")
              exists skill(name == "Grip")
              exists skill(name == "Ungrip")
              $real: skills_list()
       then
              System.out.println("DROOLS-> skill PICKPLACE is available");
              assert (new skill("PickAndPlace", new String[] {"Move", "Grip", "Ungrip"}) );
              $real.add_Skill("PickAndPlace");
end
```

Figura 5.10 – Exemplo de regra que permite ao sistema baseado em regras informar sobre a possibilidade de obter a tarefa *pickandplace*.

Como se constata pelo exemplo *PickandPlace* da Figura 5.10, as regras são auto explicativas. Os precedentes *exists* fazem-se valer do construtor "*skill (String skill_name)*" e são avaliados perante a memória de trabalho em busca de uma capacidade com nome "*skill_name*". De forma muito simplificada e com recurso a termos de programação, será realizado algo semelhante a:

workingmemory[i].name = "move"

O que traduzido em linguagem corrente pode ser traduzido como: procurar o nome pretendido numa lista de objectos. Mais abaixo, a variável "*$real*" pode ser equiparada a um apontador para o objecto do tipo *skills_list*, presente na memória de trabalho. Considerado o elo de ligação entre a memória de trabalho e o java, materializa numa lista as aptidões obtidas durante o processo de inferência. Consegue-se deste modo facultar o acesso às mesmas pela entidade agente.

A existência das três condições e da lista provoca a inserção de uma nova competência. Constituída pelos elementos condicionais que lhe deram origem é adicionada à memória de trabalho e à variável indicada.

Concluído o cálculo de novas capacidades resta registar os resultados obtidos. O comportamento - RegisterInDF consulta a lista resultante de todo o processamento anterior e informa o DF sobre as capacidades disponibilizadas pela coligação em questão.

5.3.2 Sistema de Fluxogramas

Numa outra área mas em igual período temporal do *testing_agents* é adicionado o comportamento *StartingShark*, encarregue da inicialização do executor de fluxogramas. Devido à complexidade inerente a este tipo de sistemas a interface implementada encontra-se dividida em três camadas, cada uma com funcionalidades distintas. Ao nível mais baixo é feita a gestão de directorias seus ficheiros e versões. A intermédia disponibiliza as representações existentes num ficheiro, possibilita a sua instanciação e procede à administração das já existentes, onde se inclui a manutenção das diferentes entidades e acções de comando. Mais acima, encontra-se a camada responsável por

todos os assuntos relacionados com as actividades, participantes, atributos estendidos (vulgo parâmetros) e variáveis associadas ao processo.

Em termos genéricos, o arranque do sistema é conseguido através de três métodos, enunciados no excerto de código da Figura 5.11. São eles a definição do caminho onde se encontra a base de dados, o arranque do executor propriamente dito e o estabelecimento de uma sessão ao mesmo. De agora em diante, todas as comunicações com a máquina de fluxogramas irão ocorrer por transacções.

Contudo, a requisição e distribuição de trabalho pelas entidades remotas só terá sucesso se os processos existirem na base de dados do sistema. Como tal, a interface elaborada implementa um conjunto de métodos relacionados com a selecção de ficheiros e versões, utilizados de forma automática após o arranque ou quando requisitado pelo utilizador através da interface gráfica.

```
//(1)sharkdb defined in the classpath,
LocalContextFactory.setup("sharkdb");  options!
ut = (UserTransaction)
            new InitialContext().lookup("java:comp/UserTransaction");
ut.setTransactionTimeout(50 * 60);
ut.begin();
      Shark.configure(confFilePath); //(2)starting shark
ut.commit();

ut.begin();
      WMConnectInfo wmci = new WMConnectInfo("user","","","");
      //(3)connecting to shark
      sharkc = Shark.getInstance().getSharkConnection();
      sharkc.connect(wmci);
      sh = sharkc.getSessionHandle();
ut.commit();
} catch (Throwable ex) {
                  //error closes everything
```

Figura 5.11 - Excerto de código referente à inicialização e arranque do sistema de fluxogramas.

O Fragmento de código da Figura 5.12 exibe de forma simplificada, os métodos utilizados durante a selecção automática inicial. Através dele observamos a alteração do directório raiz ao valor presente na interface gráfica, a listagem dos ficheiros nele contidos, à qual é agregada a informação sobre o estado actual de cada um com relação ao sistema, e a obtenção do identificador único associado ao ficheiro indicado na mesma interface. Ao combinar as informações anteriores e na eventualidade de o mesmo já se encontrar activo o agente irá requerer a confirmação de substituição por parte do utilizador.

```
transaction_status = XPDLRes.ChangeRepositoryDir(repositoryFilePath);
transaction_status = XPDLRes.ListXPDLsTYPE(sh);
pkgId = XPDLRes.getXPDLID(current_file);
transaction_status = XPDLRes.uploadXPDL2(sh, current_file);

if (transaction_status == NOT_ALLOWED){
        final Frame header = new Frame();
        Object[] options = {"SIM","NÃO"};
        …
        //getting the button pressed
        if (button_result == JOptionPane.YES_OPTION){
        //reload
                transaction_status = XPDLRes.updatePackage(sharkc.getSessionHandle(),
                                pkgId, current_file);
        }
}
addBehaviour(new CoalitionLeaderBehaviour(this,mt));
```

Figura 5.12 - Excerto de código referente à selecção e *loading* inicial do ficheiro de fluxogramas.

A interface gráfica referida (Figura 5.13) é útil em situações onde ocorrem erros durante o processo inicial, por exemplo devido à existência de incorrecções no XPDL, ou caso se revele necessário proceder a modificações durante o funcionamento. As operações utilizadas são semelhantes, no entanto, a interacção com o utilizador sofre algumas alterações.

Figura 5.13 – Excerto da interface destinado à definição de directorias.

Após modificação (caso necessário) dos campos *Directório* e *Ficheiro*, a interface de programação criada analisa o estado do executor e questiona o utilizador em conformidade. Entre outras, pode-lhe ser dada a opção de terminar os processos existentes ou proceder à sua actualização.

O sistema encontra-se agora pronto e à espera de pedidos por parte de entidades remotas. Ao ser requisitada uma operação o comportamento da Figura 5.14, adicionado aquando da atribuição de directorias e ficheiros, toma o controlo.

```
public CoalitionLeaderBehaviour(Agent a, MessageTemplate mt)
```

Figura 5.14 - Cabeçalho do comportamento AchieveReResponder responsável por responder ao agente requerente e, caso todos os pré-requisitos se verifiquem, dar inicio ao comportamento responsável pela execução da tarefa propriamente dita.

A sua função é responder segundo o protocolo FIPA e, caso as condições se verifiquem, disponibilizar os meios que permitam dar inicio a um novo processo.

O grupo de operações nele contido começa por analisar o pedido efectuado. Caso seja de um tipo não suportado, o sistema envia uma mensagem de recusa e entra de novo em modo de espera. Na situação contrária, resumida pelo código exposto lateralmente (Figura 5.16), é obtida uma listagem dos fluxogramas actualmente exequíveis. Se da sua conjugação com o pedido resultar algum ponto de contacto e existir disponibilidade por parte da coligação, esta procede à criação de um novo comportamento, encarregue não só de instanciar o processo bem como de gerir toda a comunicação com os agentes remotos. Terminada a sequência de acções descrita na Figura 5.15 é enviada resposta concordante à entidade responsável pelo pedido. Completa-se assim a primeira de duas fases do protocolo de interacção FIPA

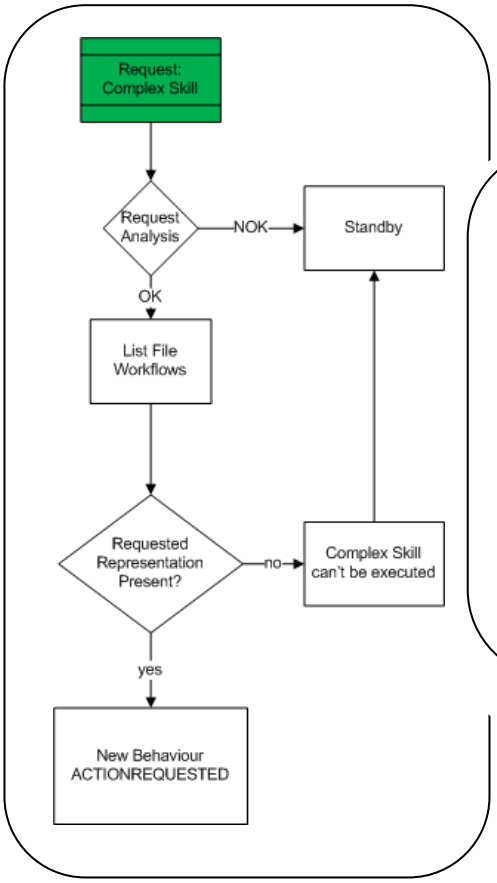

Figura 5.15 – Fluxograma do *CoalitionLeaderAgent* relativo ao comportamento *AchieveReResponder*

```
transaction_status = ProcessRes.listPKGtemplates(sk
while(it.hasNext()){
        String sk = (String)it.next();
        if(sk.equalsIgnoreCase(sc.getCoalitionRequest
                if (ProcessRes.templatefs_contains(sha
                        found = true;
                        myGui.writeResume("YES!!!!");
                        break;
                }
        }
}
if(found && !executing){
        //reply with agree to requester agent
        reply.setPerformative(ACLMessage.AGREE);
        getContentManager().fillContent(reply, new Ac
        addBehaviour(new ACTIONREQUESTED(myAgent, sc
}
```

Figura 5.16 – Código *CoalitionLeaderAgent*

. O cabeçalho da Figura 5.17 refere-se ao comportamento recentemente criado. A implementação do seu método acção constitui parte importante do sistema final. Por esse motivo, encontra-se dividido em quatro grandes áreas com propósitos tão distintos como seja o tratamento de erros, instanciação do processo, obtenção do item de trabalho a executar, junto com os seus diversos elementos, e o preenchimento e entrega dos parâmetros necessários.

```
public ACTIONREQUESTED (Agent a,String RequestName, List members, List
coalition_args, ACLMessage toSend)
```

Figura 5.17 - Cabeçalho do comportamento CyclicBehaviour responsável por instanciar o processo passado pelo parâmetro "RequestName", distribuir as suas tarefas e proceder à recuperação de possíveis erros.

O esquema da Figura 5.18 sintetiza parte da implementação actualmente em funcionamento. Por se encontrar dentro de um mecanismo cíclico, a sua execução é repetida até surgir indicação em contrário. O comportamento em questão pode ser terminado quer por comunicação de conclusão do processo, quer pela ocorrência de erro. A cada iteração o valor da variável *status* informa sobre a existência de problemas e, caso necessário, despoleta o término de todas as actividades e processos em execução através das funções *TerminateALLActivity_global e terminateALLprocess*. Para além disso, envia mensagem de *falha* à entidade responsável pelo pedido efectuado e procede às reinicializações necessárias para colocar o sistema em modo *pronto* - *ready*. Estado idêntico é estabelecido quando a análise do processo o der como concluído. Nesta situação, a coligação vale-se do equivalente aos resultados de uma função e através da *LGFormalParameters_END* obtém os parâmetros formais, limpa o registo de actividades associadas ao processo com *TerminateALLActivity_process e terminateProcess*, e envia uma mensagem com alguns dos valores retornados, a informar do sucesso da operação.

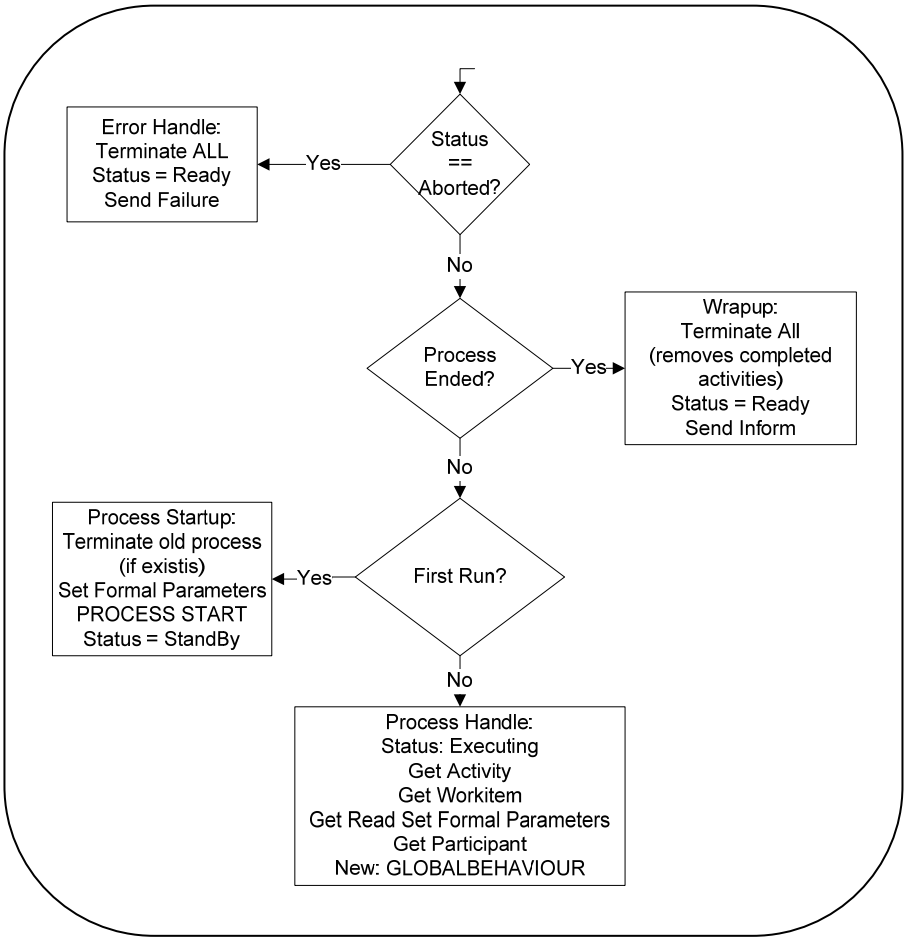

Figura 5.18 – Fluxograma descritivo do comportamento *ActionRequested*

Caso sejam ultrapassadas as primeiras verificações, fica por determinar em qual das fases ainda restantes se encontra o processo a ser avaliado. Na primeira iteração a ele relativa, a aplicação entra na secção de configuração. Nela, a função *startprocess* dá início à execução mas apenas após atribuição dos valores iniciais. Já referidos como parâmetros formais, têm de ser definidos com recurso à função *setALLFormalParameters* pois tal como numa função são necessários dados sobre os quais efectuar cálculos. Contudo, apesar de incluídos no pedido do agente remoto, na associação entre estes e os parâmetros do fluxograma, existe processamento de termos relacionados de modo a disponibilizar flexibilidade ao nível da linguagem utilizada.

O descarte de todas as situações excepcionais irá traduzir-se, com recurso às funções *getFirstProcessAct e getFirstProcessWorkitem*, na obtenção do trabalho a efectuar. No entanto o novo comportamento, responsável pela sua entrega, só poderá ser criado e tomar controlo quando os seus elementos estiverem correctamente preenchidos. Por esse motivo foram implementadas a *EA_View_Values, a GetVartoUPDATE e a LDGActivityparticipantName* que junto com processamento de termos seleccionam o agente correcto, mediante indicação do nome num todo ou em parte, o tipo de acção e o pedido pretendido, e a lista de parâmetros a ele associados.

O comportamento em questão, denominado de *GlobalBehaviour*, é instanciado para todos os itens de trabalho cuja concretização implique um processo comunicativo. Na sua implementação encontramos a versão gráfica do protocolo FIPA – contract net - com intuito de permitir a personalização de operações a efectuar, consoante a ou as respostas recebidas do pedido de trabalho ao agente remoto. Agora que o componente responsável pela comunicação com o agente remoto se encontra activo vale-se da atribuição de valores antes efectuada. Utiliza-os em funções do tipo *findedClass.newInstance()* e *Methods_TypeRequested[i].invoke(…)*, necessárias na instanciação e preenchimento dinâmico do objecto apropriado ao tipo de pedido a efectuar.

```
myAgent.getContentManager().fillContent(request,…);
messages.add(request);
ActivityRes.set_EA_UPDATE_Value(…);
```

Figura 5.19 – Excerto de código referente ao envio do pedido de operação a um agente remoto seguido pelo reportar dessa situação ao sistema de fluxogramas.

Esta sequência de operações fica completa com a expedição da mensagem e o envio do relatório das operações ao executor de fluxogramas (Figura 5.19).

Na Figura 5.20, encontram-se esquematizadas as operações anteriores e a situação excepcional, onde não foi possível encontrar a classe referente ao tipo de acção pretendida. Condição que após ser comunicada ao executor origina o término do *GlobalBehaviour* e a tomada de outras decisões, relativas à próxima acção ou medida a tomar. Em último lugar, mais abaixo na mesma figura, são também expostas as operações a executar face a cada resposta passível de ser recebida.

As diversas alternativas, apenas consideradas se for recebida resposta do agente remoto, diferem essencialmente nos Atributos Estendidos (EA[44]) fundamentais no cálculo da próxima tarefa a executar. A sua obtenção é o resultado das funções *getFirstProcessWorkitem* e *GetVartoUPDATE_VIEW* já conhecidas e utilizadas. Determinada a lista de variáveis a actualizar fica por efectuar a equivalência entre estas e os dados recebidos, e através do *Set_EA_UPDATE_Values* proceder ao seu preenchimento. Efectuadas as operações anteriores para as diversas respostas recebidas a interacção remota é dada como concluída e logo também o comportamento respectivo. Perante a sequência e valores dessas mesmas respostas, reflectivas nos EA, o *GlobalBehaviour* irá ser o responsável pelo término ou continuidade do processo, com posterior selecção de actividade pelo *ActionRequested*.

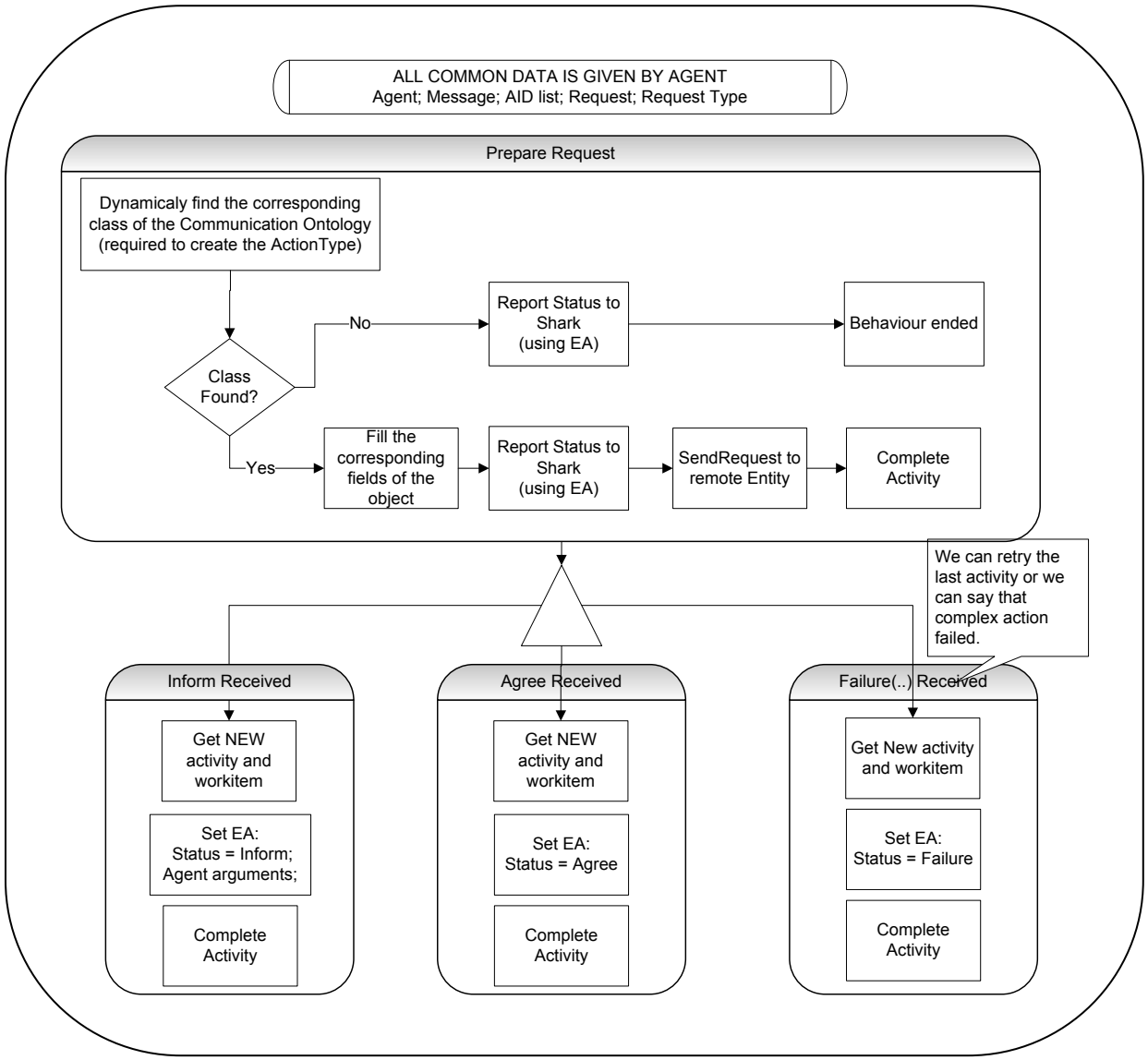

Figura 5.20- *Fluxograma relativo ao comportamento GlobalBehaviour*

[44] Extended Attributes

O Estado mais comum que se pretende obter é conseguido após recepção das mensagens *agree* e *inform* respectivamente. Nessa condição, o comportamento *GlobalBehaviour* é dado como concluído, enquanto o *ActionRequested* toma de novo a liderança, a fim de seleccionar uma nova actividade a realizar. A sequência de operações descrita será repetida até o processo em si ser dado como concluído, altura marcada pela entrada do sistema em modo de *WrapUp*, descrito antes.

Efectuados os esclarecimentos relativos à construção de regras e combinação dos diversos componentes, a lacuna referente à definição de processos permanece por elucidar. Embora a linguagem XPDL permita a construção de processos num qualquer editor, visto estar-se perante código XML estendido e orientado a processos, a operação fica facilitada caso o utilizador se sirva das alternativas gráficas disponíveis. Estas, em adição, a permitirem a construção por blocos e sub-blocos, disponibilizam menus de contexto adaptados a cada situação, facilitam a definição de variáveis, parâmetros e participantes, esclarecem sobre a hierarquização actual e efectuam a verificação de erros.

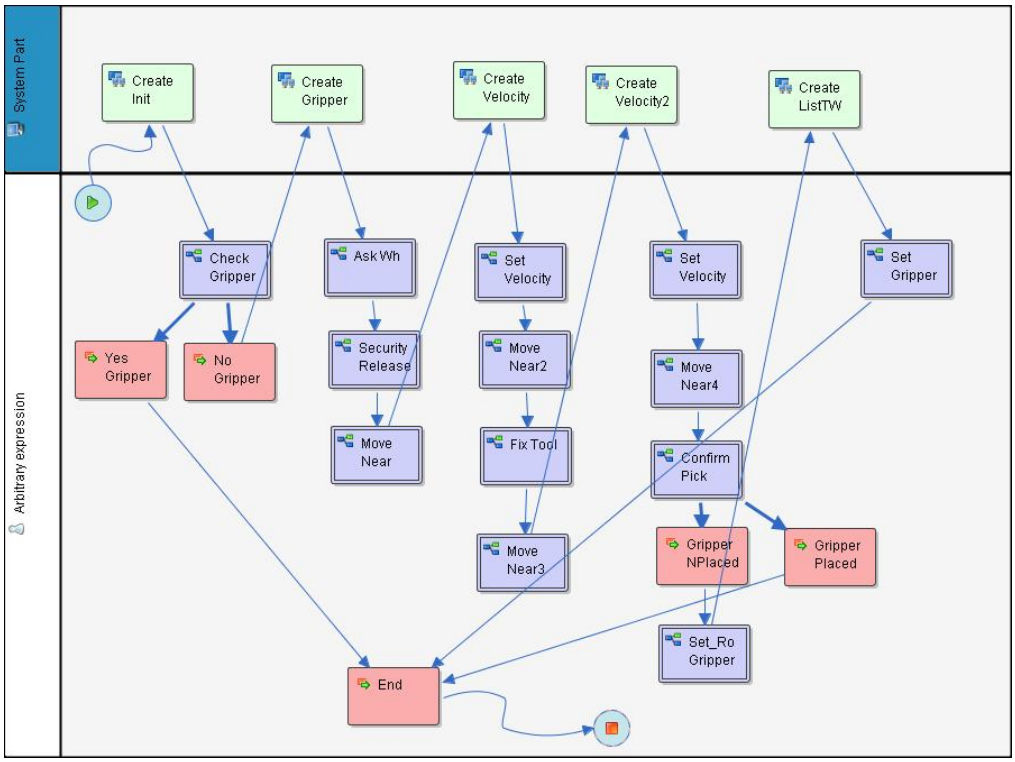

Figura 5.21- Exemplo de fluxograma relativo à actividade *PlaceTool*

O fluxograma descrito acima (Figura 5.21) expõe um possível conjunto de operações, neste caso destinadas à realização da tarefa PlaceTool. A complexidade que se lhe supõe estar associada é enganadora pois a sua construção ou modificação segue um conjunto de passos simples.

Considere-se a situação onde uma entidade competente pretende adicionar a actividade "*Close Tool"* antes do "*Move near3"*. Uma possível sequência de operações a realizar seria:

- Através do menu de ferramentas criar uma nova actividade;

- Ir ao menu de contexto e aceder às suas propriedades;
- No tabulador *SubFlow* indicar qual é o agente a que se destina a acção;
- Preencher os campos necessários à sua execução:

    Request: “GRIP”

    Model: “PGN64”

    Args: null

    Ae: “ComponentsOntology.GripperAction”

- Efectuar a conexão da nova actividade, por selecção da função de conectividade;
- Realizar uma verificação de erros e salvar o ficheiro.

O último passo prende-se com o carregamento do ficheiro no executor, situação já documentada no decorrer deste capítulo.

5.4 Estudo de um Caso Particular

Considere-se agora a situação onde se procedeu à definição do processo *PickandPlace*, muito comum em processos industriais (Figura 5.22).

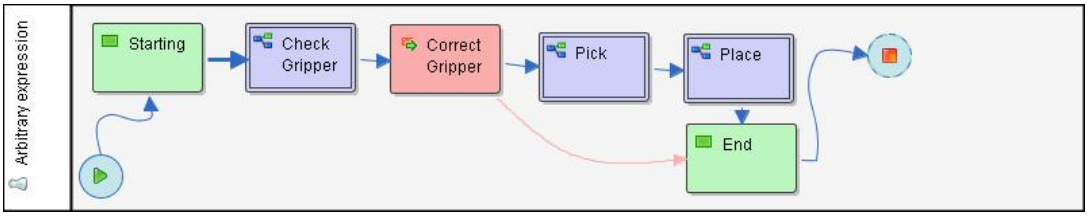

Figura 5.22 – Definição do processo *PickandPlace*

Em situação normal alguns dos agentes da célula estariam a proceder ao encaminhamento da palete por entre os diversos *conveyors* período no qual o executor de fluxogramas da coligação, que se encontra no caminho, permanece em modo de espera. Não obstante o seu sistema de regras encontrar-se-ia a ser executado em intervalos de tempo regulares. A chegada de uma palete à área de trabalho da coligação culmina na entrega de um pedido *PickandPlace*, por parte desta. Perante esta situação uma possível sequência de operações para a coligação seria dada por (sequência de operações sintetizada na Figura 5.23):

1- O comportamento *CoalitionLeaderBehaviour* recebe e verifica o pedido;
2- Confirma-se a existência do mesmo nos resultados disponibilizados pelo sistema de regras;
3- Encontrada correspondência, efectua-se uma procura pela definição do processo no executor de fluxogramas;
4- Encontrada a definição do processo, e como existe disponibilidade por parte do agente, cria-se um novo comportamento destinado a efectuar a sua gestão. Informa-se também a palete, entidade responsável pelo pedido, que o mesmo se encontra a ser executado;
5- O *CoalitionLeaderBehaviour* é dado como terminado;

6- O novo comportamento *ActionRequested* começa por proceder à instanciação, atribuição de valores necessários e arranque do processo seleccionado anteriormente;
7- Devido à sua natureza a iteração seguinte determina qual a próxima actividade a realizar;
8- A operação anterior retorna como resultado a actividade *CheckGripper*;
9- Como se está perante uma actividade cuja implementação é dada pela representação do protocolo FIPA é criado uma instância do *GlobalBehaviour*;
10- Este novo comportamento, encarregue da comunicação, começa por preencher os campos do objecto adequado ao tipo de entidade com quem se pretende estabelecer comunicação. Na situação em questão, pretende-se enviar a mensagem ao robô, logo o objecto a ser preenchido e enviado denomina-se *InformationAction*;
11- *O GlobalBehaviour* permanece em modo de espera até ser recebida resposta. Altura em que se procede à recuperação dos parâmetros nela presentes e entrega dos mesmos ao executor de fluxogramas;
12- O executor perante os dados recebidos e devido ao tipo de acção requerida dá por concluído o *GlobalBehaviour*;
13- Perante esta situação o *ActionRequested* toma de novo o controlo e consoante os resultados determina a próxima acção a executar. Com ela repete-se todo o mecanismo a partir do passo 8. Situação que se repete até o processo ser dado como concluído ou pela ocorrência de erro;
14- No final, é efectuada a recuperação de resultados, a devolver na segunda mensagem de resposta à palete. O agente remoto e a coligação permanecem assim em conformidade com o protocolo FIPA;
15- O *ActionRequest* é dado como concluído;
16- Todo o agente de coligação entra agora em modo de espera;

Atente-se agora no caso em que a coligação é enriquecida com mais alguns elementos agentificados, nomeadamente um sistema de suporte de ferramentas e uma nova ferramenta de manuseamento para o robot. Perante esta situação novas capacidades poderão estar disponíveis. Na versão primitiva tais ocorrências iriam implicar a reprogramação da coligação de modo a que novas *skills* oferecidas possam ser utilizadas. Na nova coligação o processo referido é substituído pela modificação do ficheiro de regras e descrição de fluxogramas, isto no caso de os mesmos não deterem já informação suficiente que lhes permita lidar com a nova condição do sistema (processo descrito nos subcapítulos anteriores).

Garantida a validade dos diferentes ficheiros resta proceder à sua transferência para a coligação em execução. Após essa situação e para o caso concreto apresentado serão agora detectadas as capacidades de *PickandPlace* e *ExchangeGripper*.

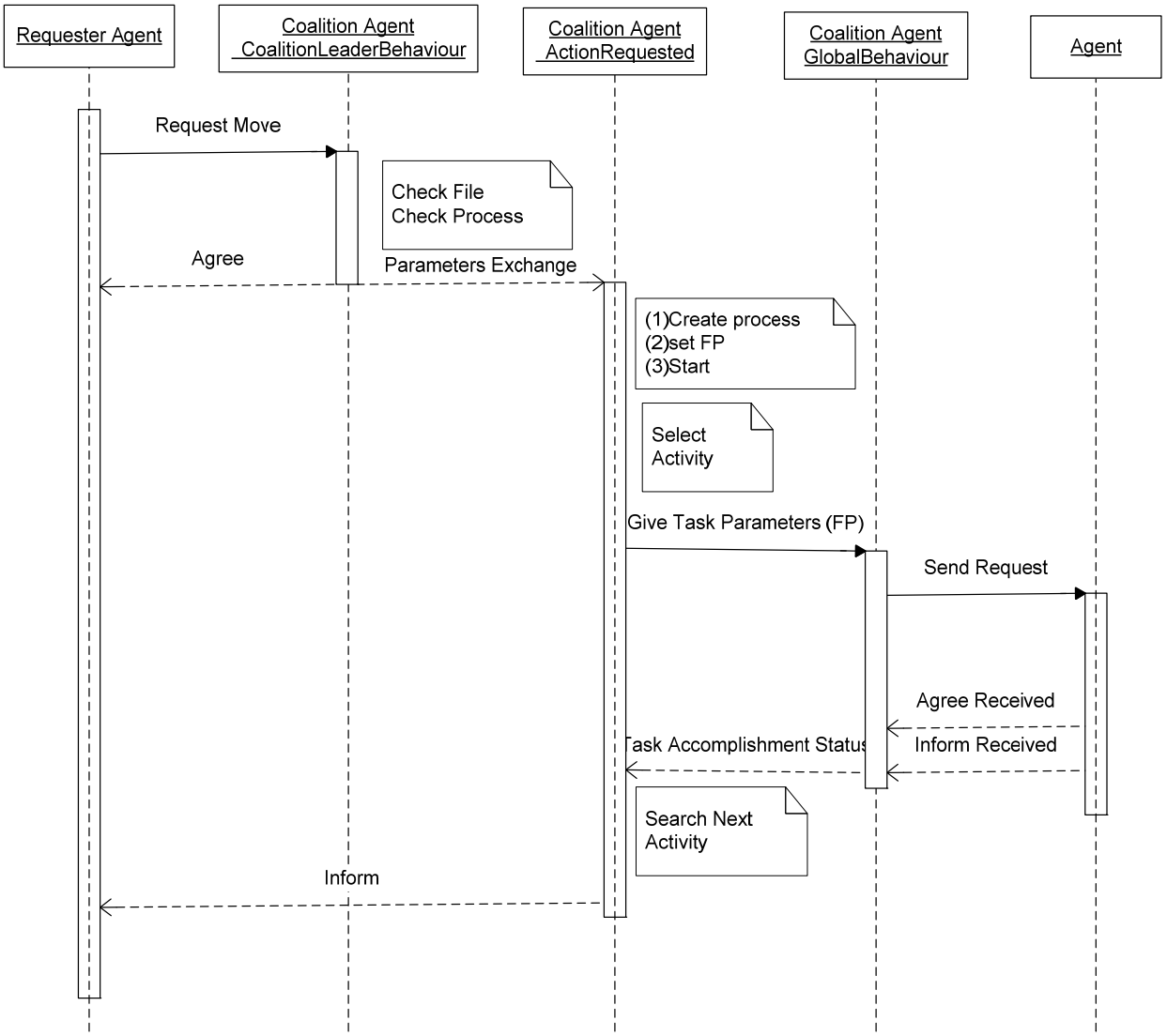

Figura 5.23 - Diagrama com as diversas mensagens transferidas e operações a decorrer, em cada um dos agentes, quando um agente efectua o pedido de uma operação.

6 Epílogo

6.1 Conclusão

O Agente de Coligação dinâmico foi esquematizado, implementado e testado com sucesso. O utilizador comum pode proceder à definição e *loading* de regras e processos de forma rápida e intuitiva. Com intuito de oferecer outro nível de versatilidade, é também possível efectuar alterações dinâmicas sobre as definições anteriores. A aplicação resultante elimina a necessidade de recorrer a programação e oferece interfaces gráficas quer na especificação dos requisitos para a execução quer na própria definição do processo. No entanto, para o utilizador mais experiente é sempre possível recorrer ao método mais comum e antigo como seja a programação.

A selecção e integração do sistema baseado em regras e do sistema de *fluxogramas*, no agente de coligação, foi completamente realizada. Com o mesmo desfecho, o agente de coligação foi refeito de modo a poder comportar as novas funcionalidades.

Com recurso a aplicações externas foram efectuadas diversas especificações de regras e processos a utilizar durante o ciclo de vida do agente. A definição das acções mais comuns e regras correspondentes servem não só de exemplo, como de fundação para definições futuras.

O coordenador criado foi testado na célula de manufactura experimental, presente na Uninova, onde cada entidade física se encontra abstraída por um Agente. Com ele foi possível determinar e executar as tarefas complexas *PickandPlace* e *ExchangeGripper* com sucesso. Face a problemas da coligação constatou-se que é seguida uma das sequências de erro pré-elaborada nas definições de processos. Caso a mesma não se encontre definida procede-se ao término de todos os mecanismos associados à instância do processo.

Por último, é possível proceder a alterações nos processos durante a execução desde que realizadas atempadamente. Essas modificações são sempre simples e fáceis de realizar graças às aplicações introduzidas.

Algumas aproximações iniciais foram apresentadas. A sua implementação permitiu verificar o acréscimo de complexidade, a ausência de funcionalidades ou problemas de comunicação, factores importantes na posterior fase de definição da arquitectura actual.

A existência de múltiplos *standards* principalmente na área de definição de processos gráficos criou alguma entropia no entanto acabou por se optar pelo XPDL, linguagem muito similar ao XML, esta última difundida universalmente. A decisão anterior simplificou a posterior escolha da aplicação a utilizar, uma vez que existia menos um factor a pesar na decisão.

A coligação criada e as aplicações externas têm como publico alvo os utilizadores sem conhecimentos específicos de programação, no entanto também está apta a ser utilizada pelos implementadores de sistemas. Explora as capacidades das diversas aplicações integradas, de forma a não ser necessária qualquer intervenção para além da indicação do ficheiro de regras e processos a utilizar. Graças à natureza do sistema criado é possibilitada uma maior personalização e controlo dos erros passíveis de surgir durante e execução.

No interior da coligação existe a implementação de um agente modificado de modo a comportar um sistema baseado em regras e um sistema executor de fluxogramas. Não obstante, do ponto de vista externo, estamos perante um agente similar a qualquer outro que publica os seus serviços consoante as suas possibilidades.

A aproximação actual oferece um grande número de vantagens sobre a implementação clássica. Entre elas a possibilidade de definir processos de forma gráfica e modelar regras de forma intuitiva, com recurso inclusive a linguagem de domínio. A alteração dos processos e regras durante a execução, a capacidade de lidar com situações não previstas durante a fase de especificação e o suporte a diversas falhas, também se encontra contemplado.

No final, o agente resultante aumenta o nível de agilidade e dinamismo do ambiente fabril ao providenciar suporte à reengenharia e modificações do produto. Como tal, a anterior fase de programação encontra-se substituída pela configuração.

A plataforma de agentes já existente apesar de útil e completa trouxe algumas limitações em termos de verificação de parâmetros e formatos de dados. As situações anteriores têm como causa o facto de o primeiro protótipo da coligação ter sido desenvolvido em período idêntico ao resto da estrutura e assim a sua adaptação era total. Esta poderá ser uma área onde podem ocorrer melhorias na restante rede.

O desenvolvimento de *hardware* habilitado a executar de forma nativa os agentes JADE irá aumentar a aceitação do conceito, permitir a inclusão dos mesmos em todos os elementos da linha de produção e possibilitar a obtenção de respostas de forma mais rápida.

Mais uma vez e em concordância com outros projectos efectuados, foram verificados alguns atrasos na execução de acções de *hardware*. A situação anterior leva a concluir que os agentes não são apropriados em operações de baixo nível, pelo menos até que o grau de processamento exigido pelo seu processo de decisão não seja comportado. No enquadramento actual, os agentes devem ficar circunscritos às decisões de alto nível onde não existem restrições de tempo elevadas.

A implementação que serviu de base foi analisada e pequenas falhas foram detectadas. São necessárias bastantes configurações para que os agentes possam funcionar em computadores

diferentes dos inicialmente utilizados. Em parte das aplicações adicionais requeridas é recomendável recorrer a versões e directorias idênticas de modo a evitar erros na inicialização e possibilitar a correcta obtenção de informação do lado dos agentes. A ontologia de comunicação apesar de comum não se encontra sujeita a nenhum *standard* e não possui nenhum método de partilha automático. Surgem, deste modo, diferentes versões sendo estas uma porta para a ocorrência de problemas de comunicação. Apesar de ser apresentado como um sistema descentralizado existem módulos, nomeadamente o DF, cujo funcionamento após diversas actualizações e revisões ainda continua a ser, na sua essência, centralizado. Por último, face a uma falha ou indisponibilidade na realização de uma operação segue-se a repetição do pedido de forma indefinida (pelo agente requerente).

Durante a execução do projecto, foram surgindo algumas aproximações com alguns pontos de interesse, nomeadamente o *WADE* [65] e o *Drools Flow* [212] porém para o primeiro existem limitações ao nível do carregamento dos processos. Na outra alternativa, a funcionalidade do sistema de regras é diferente da pretendida.

O trabalho apresentado nesta tese encontra-se publicado nos *proceedings* do simpósio internacional em electrónica industrial do ano de 2011, como nome - Dynamic Skills Composition and Execution in a Multi-agent Manufacturing System.

6.2 Trabalho Futuro

Durante o desenvolvimento da aplicação foram emergindo novas ideias. De entre as diversas evoluções enunciam-se as mais importantes, algumas das quais podem ser pontos de partida para trabalhos futuros.

- Conjugar os *webservices* com os agentes de modo conseguir obter uma plataforma ainda mais distribuída e directa[45];

- Proceder às diversas melhorias na estrutura de agentes existente;

- Integrar e testar de forma efectiva o armazém de produtos associado à célula;

- Alterar a implementação actual de modo a ser possível escolher entre o modo de comunicação FIPA *standard* e o modo onde se definem operações específicas para uma das respostas recebidas (*Not understand, agree, inform, fail* etc.) que se encontra actualmente implementado;

- Elaboração de uma aplicação autónoma construtora de sequências de operações através de desenhos CAD;

[45] http://processdevelopments.blogspot.com/2007/04/bpel-compared-to-jpdl.html

6.3 Referências Bibliográficas

1. Jovane, F., *Redesigning manufacturing: Manufuture.* Production Engineering and Computers, 2004. **6**(7): p. 1-6.
2. Comission, E. *MANUFUTURE Platform*. Available from: http://www.manufuture.org.
3. Geyer, A., et al., *the future of manufacturing in europe 2010 2015 the challenge for sustainability*. 2003, European Commission.
4. Cândido, G.M. and F.N.R. Feijão, *Novalex Shop Floor Cell Agentification.* 2006.
5. Setchi, R.M. and N. Lagos. *Reconfigurability and reconfigurable manufacturing systems: state-of-the-art review*. in *INDIN '04. 2nd International Conference on Industrial Informatics*. 2004.
6. Doll, W.J. and M.A. Vonderembse, *The evolution of manufacturing systems: Towards the post-industrial enterprise.* Omega, 1991. **19**(5): p. 401-411.
7. Masi, D.D., *A Sociedade Pós-Industrial*, ed. E. SENAC. 2000, São Paulo.
8. Saadat, M., *Changes and disturbances in manufacturing systems: A comparison of emerging concepts.* 2008 WORLD AUTOMATION CONGRESS PROCEEDINGS, VOLS 1-3, 2008: p. 555-560.
9. Hitomi, K., *Manufacturing systems engineering: a unified approach to manufacturing technology, production management, and industrial economics*. 2nd ed, ed. T.a. Francis. 1996: CRC.
10. Womack, J.P., D.T. Jones, and D. Roos, *The Machine that Changed the World: The Story of Lean Production*. 1990, New York: Rawson Associates.
11. Leitão, P., *An agile and adaptive holonic architecture for manufacturing control*. 2004.
12. Oliveira, J.A.B.d., *Coalition based approach for shop floor agility – a multiagent approach*, in *DI*. 2003, FCT-UNL.
13. Mehrabi, M.G., *Reconfigurable manufacturing systems: Key to future manufacturing.* JOURNAL OF INTELLIGENT MANUFACTURING, 2000. **11**(4): p. 403-419.
14. Ueda, K., *A Concept for Bionic Manufacturing Systems Based on DNA-type Information*, in *Proceedings of the IFIP TC5 / WG5.3 Eight International PROLAMAT Conference on Human Aspects in Computer Integrated Manufacturing*. 1992, North-Holland Publishing Co. p. 853-863.
15. Gou, L., P.B. Luh, and Y. Kyoya, *Holonic manufacturing scheduling: architecture, cooperation mechanism, and implementation.* Computers in Industry, 1998. **37**(3): p. 213-231.
16. Bussmann, S. and D.C. McFarlane, *Rationales for Holonic Manufacturing Control*, in *In Proc. of Second Int. Workshop on Intelligent Manufacturing Systems*. 1999. p. 17--7.
17. Van Brussel, H., et al., *Reference architecture for holonic manufacturing systems: PROSA.* Computers in Industry, 1998. **37**(3): p. 255-274.
18. Babiceanu, R. and F. Chen, *Development and Applications of Holonic Manufacturing Systems: A Survey.* JOURNAL OF INTELLIGENT MANUFACTURING, 2006. **17**(1): p. 111-131.
19. Koren, Y., et al., *Reconfigurable Manufacturing Systems.* CIRP Annals - Manufacturing Technology, 1999. **48**(2): p. 527-540.
20. Mehrabi, M., A. Ulsoy, and Y. Koren, *Reconfigurable manufacturing systems and their enabling technologies.* International Journal of Manufacturing Technology and Management, 2000. **1**(1): p. 114-131.
21. Onori, M., H. Alsterman, and J. Barata. *An architecture development approach for evolvable assembly systems*. in *Assembly and Task Planning: From Nano to Macro Assembly and Manufacturing, 2005. (ISATP 2005). The 6th IEEE International Symposium on*. 2005.
22. Barata, J., P. Santana, and M. Onori, *Evolvable Assembly Systems: A Development Roadmap*, in *Symposium on Information Control Problems in Manufacturing*, IFAC, Editor. 2006: Saint-Etienne, France.
23. Onori, M. *Evolvable Assembly Systems-A New Paradigm?* in *International Symposium on Robotics*. 2002.

24. Frei, R., J. Barata, and M. Onori. *Evolvable Production Systems Context and Implications*. in *Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on*. 2007.
25. Barata, J., et al., *Evolvable Production Systems: Enabling Research Domains*, in *2nd International Conference on Changeable, Agile, Reconfigurable, and Virtual Production*. 2007: Canadá.
26. Marik, V. *Holons & Agents: Recent Development and Mutual Impacts*. 2001.
27. Marik, V. and D. McFarlane, *Industrial adoption of agent-based technologies.* Intelligent Systems, IEEE, 2005. **20**(1): p. 27-35.
28. Hall, K.H., *Experience with holonic and agent-based control systems and their adoption by industry.* HOLONIC AND MULTI-AGENT SYSTEMS FOR MANUFACTURING, PROCEEDINGS, 2005. **3593**: p. 1-10.
29. Leitão, P. and F. Restivo, *Implementation of a holonic control system in a flexible manufacturing system*. 2008.
30. Brennan, R.W., *Developments in dynamic and intelligent reconfiguration of industrial automation.* Computers in Industry, 2008. **59**(6): p. 533-547.
31. Leitão, P., A.W. Colombo, and J.M. Mendes, *Smooth Migration from the Virtual Design to the Real Manufacturing Control*. 2009.
32. Schild, K. and S. Bussmann, *SELF-ORGANIZATION IN MANUFACTURING OPERATIONS.* Communications of the ACM, 2007. **50**(12): p. 74-79.
33. ElMaraghy, H.A., *Flexible and reconfigurable manufacturing systems paradigms.* International Journal of Flexible Manufacturing Systems, 2005. **17**(4): p. 261-276.
34. Weiming, S., S.Y.T. Lang, and W. Lihui, *iShopFloor: an Internet-enabled agent-based intelligent shop floor.* Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 2005. **35**(3): p. 371-381.
35. Botti, V., et al., *The Artis Agent Architecture: Modelling Agents in Hard Real-Time Environments*, in *Multi-Agent System Engineering*, F. Garijo and M. Boman, Editors. 1999, Springer Berlin / Heidelberg. p. 63-76.
36. PROMISE, P. 2010; Available from: http://www.pabadis-promise.org/.
37. Rooker, M.N., et al. *Adaptive and Reconfigurable control framework for the responsive factory*. in *Industrial Informatics, 2009. INDIN 2009. 7th IEEE International Conference on*. 2009.
38. Leitão, P. and F. Restivo. *A framework for Distributed Manufacturing Applications*. in *In proceedings of the 2000 Advanced Summer Institute (AIS)*. 2000.
39. Leitao, P., *Self-Organization in Manufacturing Systems: Challenges and Opportunities.* Self-Adaptive and Self-Organizing Systems Workshops, 2008. SASOW 2008. Second IEEE International Conference on, 2008: p. 174-179.
40. Bi, Z.M., et al., *Reconfigurable manufacturing systems: the state of the art.* International Journal of Production Research, 2008. **46**(4): p. 967 - 992.
41. Christo, C. and C. Cardeira. *Trends in Intelligent Manufacturing Systems*. in *Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on*. 2007.
42. Tharumarajah, A., *Comparison of the bionic, fractal and holonic manufacturing system concepts.* International Journal of Computer Integrated Manufacturing, 1996. **9**(3): p. 217 - 226.
43. Tharumarajah, A., A.J. Wells, and L. Nemes, *Comparison of emerging manufacturing concepts.* 1998 IEEE International Conference on Systems, Man, and Cybernetics, 1998. **1**: p. 325-331 vol.1.
44. Sihn, W. *Re-engineering through Fractal Structures*. in *IFIP TC 5 WG 5.7 Working Conference on Re-engineering the Enterprise*. 1995. Galway, Ireland.
45. Steele, J.E., *Bionic design of intelligent systems*. 1977, Wright State University.
46. Okino, N. *Bionic manufacturing systems*. in *bionic manufacturing systems*. 1989. Elsevier: Elsevier.
47. Okino, N. *A prototyping of bionic manufacturing system*. in *International Conference on Objectoriented Manufacturing Systems*. 1992. Calgary, Alberta,.
48. Okino, N. *Bionic manufacturing system*. in *Flexible Manufacturing Systems Past-Present-Future*. 1993: CIRP.

49. Koestler, A., *The Ghost in the machine*. 1967, London: Arcana Books.
50. IMS. *IMS Consortium*. Available from: www.ims.org.
51. Monostori, L., J. Váncza, and S.R.T. Kumara, *Agent-Based Systems for Manufacturing.* CIRP Annals - Manufacturing Technology, 2006. **55**(2): p. 697-720.
52. Bussmann, S. *An agent-oriented architecture for holonic manufacturing control*. in *IMS 1998*. 1998. Lausanne, Switzerland.
53. Bussmann, S. and K. Schild. *An agent-based approach to the control of flexible production systems*. in *Proceedings. 2001 8th IEEE International Conference on Emerging Technologies and Factory Automation, 2001.* . 2001.
54. Jennings, N.R., K. Sycara, and M. Wooldridge, *A Roadmap of Agent Research and Development.* Autonomous Agents and Multi-Agent Systems, 1998. **1**(1): p. 7-38.
55. Hewitt, C., *Viewing control structures as patterns of passing messages.* Artificial Intelligence, 1977. **8**(3): p. 323-364.
56. Russel, S. and P. Norvig, *Artificial Intelligence, A Modern Approach*. 1995: Prentice-Hall.
57. Wooldridge, M. and N.R. Jennings, *Intelligent agents: theory and practice.* The Knowledge Engineering Review, 1995. **10**(02): p. 115-152.
58. Luck, M., P. McBurney, and C. Preist, *Agent Technology: Enabling Next Generation Computing (A Roadmap for Agent Based Computing)*. 2003: AgentLink.
59. Franklin, S. and A. Graesser, *Is It an agent, or just a program?: A taxonomy for autonomous agents*, in *Intelligent Agents III Agent Theories, Architectures, and Languages*, J. Müller, M. Wooldridge, and N. Jennings, Editors. 1997, Springer Berlin / Heidelberg. p. 21-35.
60. Jennings, N. and M. Wooldridge. *Applications of intelligent agents*. in *Agent technology: Foundations, applications and markets*. 1998: Springer.
61. Colombo, A.W., R. Schoop, and R. Neubert, *An agent-based intelligent control platform for industrial holonic manufacturing systems.* Industrial Electronics, IEEE Transactions on, 2005. **53**(1): p. 322-337.
62. Wooldridge, M. and N. Jennings, *Agent theories, architectures, and languages: A survey*, in *Intelligent Agents*, M. Wooldridge and N. Jennings, Editors. 1995, Springer Berlin / Heidelberg. p. 1-39.
63. Ferber, J., *Multi-Agent System: An Introduction to Distributed Artificial Intelligence*, ed. A.W. Longman. 1999.
64. Wooldridge, M., *Methodologies*, in *An Introduction to Multi-Agent Systems*. 2002, John wiley & Sons: West Sussex, UK. p. 225-244.
65. JADE. *The Foundation for Intelligent Physical Agents*. 2009 [cited 2009; Available from: http://jade.tilab.com/.
66. Sycara, K., et al., *The RETSINA MAS Infrastructure.* Autonomous Agents and Multi-Agent Systems, 2003. **7**(1): p. 29-48.
67. FIPA. *The Foundation for Intelligent Physical Agents* 2002 [cited 2009; Available from: http://www.fipa.org.
68. AgentBuilder. *AgentBuilder*. 2009 2009]; Available from: http://www.agentbuilder.com/.
69. Wooldridge, M., *A history Lesson*, in *An Introduction to Multi-Agent Systems*. 2002, John wiley & Sons: West Sussex, UK. p. 303-315.
70. Tang, H.P. and T.N. Wong, *Reactive multi-agent system for assembly cell control.* Robotics and Computer-Integrated Manufacturing, 2005. **21**(2): p. 87-98.
71. Lee, J.-H. and C.-O. Kim, *Multi-agent systems applications in manufacturing systems and supply chain management: a review paper.* International Journal of Production Research, 2008. **46**(1): p. 233 - 265.
72. Barata, J. and M. Onori. *Evolvable Assembly and Exploiting Emergent Behaviour*. in *Industrial Electronics, 2006 IEEE International Symposium on*. 2006.
73. Ueda, K., et al., *Emergent Synthesis Methodologies for Manufacturing.* CIRP Annals - Manufacturing Technology, 2001. **50**(2): p. 535-551.
74. Leitao, P., *Holonic Rationale and Bio-inspiration on Design of Complex Emergent and Evolvable Systems*, in *Transactions on Large-Scale Data- and Knowledge-Centered Systems I*. 2009, Springer-Verlag. p. 243-266.

75. Nwana, H.S., *Software agents: an overview.* The Knowledge Engineering Review, 1996. **11**(03): p. 205-244.
76. Jennings, N.R. and S. Bussmann, *Agent-based control systems: Why are they suited to engineering complex systems?* Control Systems Magazine, IEEE, 2003. **23**(3): p. 61-73.
77. Gruber, T.R., *A translation approach to portable ontology specifications.* Knowl. Acquis., 1993. **5**(2): p. 199-220.
78. Smith, R.G., *The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver.* Computers, IEEE Transactions on, 1980. **C-29**(12): p. 1104-1113.
79. Linthicum, D.S., *Enterprise application integration*. 2000: Addison-Wesley Longman Ltd. 377.
80. Camarinha-Matos, L.M., et al., *Towards an architecture for virtual enterprises.* JOURNAL OF INTELLIGENT MANUFACTURING, 1998. **9**(2): p. 189-199.
81. Bouchoul, F. and M. Mostefai, *From e-Manufacturing to M-Manufacturing*.
82. ZEUS. *ZEUS*. 2000 [cited 2006; Available from: http://labs.bt.com/projects/agents/zeus.
83. Cougaar. *Cougaar - Cognitive Agent Architecture*. 2000 [cited 2009; Available from: http://www.cougaar.org/.
84. Pereira, C.E. and L. Carro, *Distributed real-time embedded systems: Recent advances, future trends and their impact on manufacturing plant control.* Annual Reviews in Control, 2007. **31**(1): p. 81-92.
85. Bussmann, S., N.R. Jennings, and M. Wooldridge, *Multiagent systems for manufacturing control: a design methodology*. 2004, New York: Springer-Verlag.
86. Tichy, P., et al., *Multiagent technology for fault tolerance and flexible control.* Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 2006. **36**(5): p. 700-704.
87. Pěchouček, M. and V. Mařík, *Industrial deployment of multi-agent technologies: review and selected case studies.* Autonomous Agents and Multi-Agent Systems, 2008. **17**(3): p. 397-431.
88. Markfort, D. and H. Kühnle. *The Re-configurable Enterprise - an appearing paradigm of future manufacturing*. 1999.
89. Leitão, P., *Agent-based distributed manufacturing control: A state-of-the-art survey.* Engineering Applications of Artificial Intelligence, 2009. **22**(7): p. 979-991.
90. Pěchouček, M., M. Rehák, and V. Marik, *Expectations and deployment of agent technology in manufacturing and defence: case studies*, in *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*. 2005, ACM: The Netherlands. p. 100-106.
91. Bordini, R., M. Dastani, and M. Winikoff, *Current Issues in Multi-Agent Systems Development*, in *Engineering Societies in the Agents World VII*, G. O'Hare, et al., Editors. 2007, Springer Berlin / Heidelberg. p. 38-61.
92. Huhns, M.N., et al., *Research directions for service-oriented multiagent systems.* Internet Computing, IEEE, 2005. **9**(6): p. 65-70.
93. Joeris, G., *Decentralized and flexible workflow enactment based on task coordination agents*, in *in Proc. of the 2nd Int. Bi-Conference Workshop on Agent-Oriented Information Systems (iCue Publishing*. 2000, iCue Publishing: Stockholm, Sweden. p. 41--62.
94. Buhler, P.A. and J.M. Vidal, *Towards Adaptive Workflow Enactment Using Multiagent Systems.* Information Technology and Management, 2005. **6**(1): p. 61-87.
95. Stormer, H., *AWA - A flexible Agent-Workflow System*, in *Fifth International Conference on Autonomous Agents (AGENTS 2001)*. 2001: Montreal, Canada.
96. Wang, M. and H. Wang, *Intelligent Agent Supported Flexible Workflow Monitoring System*, in *Advanced Information Systems Engineering*, A. Pidduck, et al., Editors. 2006, Springer Berlin / Heidelberg. p. 787-791.
97. Jennings, N.R., et al., *Autonomous Agents for Business Process Management.* Int. Journal of Applied Artificial Intelligence, 2000. **14**(2): p. 145-189.
98. Nissen, M.E. *Supply Chain Process and Agent Design for E-Commerce*. in *33rd International Conference on System Sciences*. 2000. Hawaii.

99. Georgakopoulos, D., M. Hornick, and A. Sheth, *An overview of workflow management: From process modeling to workflow automation infrastructure.* Distributed and Parallel Databases, 1995. **3**(2): p. 119-153.
100. Mohan, C., *Tutorial: State of the Art in Workflow Management System Research and Products*, in *International Conference on Extending Database Technology*. 1996: Avignon, France.
101. Yanli, H., et al. *Flexible Workflow Driven Job Shop Manufacturing Execution and Automation Based on Multi Agent System*. in *Intelligent Agent Technology, 2006. IAT '06. IEEE/WIC/ACM International Conference on*. 2006.
102. Fleurke, M., L. Ehrler, and M. Purvis. *JBees - an adaptive and distributed framework for workflow systems*. in *In Workshop on Collaboration Agents: Autonomous Agents for Collaborative Environments (COLA)*. 2003. Halifax, Canada.
103. Vossen, G., *The WASA approach to workflow management for scientific applications.* WORKFLOW MANAGEMENT SYSTEMS AND INTEROPERABILITY, 1998. **164**: p. 145-164.
104. Negri, A., et al., *Dynamic Grid tasks composition and distribution through agents.* Concurrency and Computation: Practice and Experience, 2006. **18**(8): p. 875-885.
105. Overeinder, B., et al. *Multi-Agent Support for Internet-Scale Grid Management*. in *AISB'02 Symposium on AI and Grid Computing*. 2002.
106. Poggi, A., M. Tomaiuolo, and P. Turci. *Extending JADE for Agent Grid Applications*. in *13th IEEE International Workshops on Enabling Technologies and Infrastructure for Collaborative Enterprises (WETICE)*. 2004. Modena, Italy.
107. Bhatia, D., et al., *WebFlow – a visual programming paradigm for Web/Java based coarse grain distributed computing.* Concurrency: Practice and Experience, 1997. **9**(6): p. 555-577.
108. Verma, K., et al. *On accommodating inter service dependencies in web process flow composition*. in *In AAAI Spring Symposium on Semantic Web Services*. 2004. Palo Alto, CA.
109. Junwei, C. *GridFlow: Workflow Management for Grid Computing*. in *Third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03)*. 2003. Tokyo, Japan
110. JBoss. *JBoss Community*. [cited 2009; Available from: http://www.jboss.org/.
111. Katz, E., *A Multiple Rule Engine-Based Agent Control Architecture*, in *IEEE 6th International Conference on Intelligent Engineering Systems*. 2002. p. 2001--2001.
112. Garro, A., L. Palopoli, and F. Ricca, *Exploiting agents in e-learning and skills management context.* AI Communications, 2006. **19**(2): p. 137-154.
113. Schiefer, J., et al., *Event-driven rules for sensing and responding to business situations*, in *Proceedings of the 2007 inaugural international conference on Distributed event-based systems*. 2007, ACM: Toronto, Ontario, Canada. p. 198-205.
114. Botea, A., M. Müller, and J.S. er, *Using component abstraction for automatic generation of macro-actions*, in *In Proceedings of the International Conference on Automatic Planning and Scheduling ICAPS-04*. 2004. p. 181--190.
115. Swaminathan, A., S.A. Shaikh, and K.S. Barber, *Design of an experience-based assembly sequence planner for mechanical assemblies.* Robotica, 1998. **16**(3): p. 265-283.
116. Benfer, R.A., E.E. Brent, and L. Furbee-Losee, *Expert systems*. Quantitative Applications in the Social Sciences. Vol. 77. 1991, Newbury Park, London, New Delhi: SAGE Publications.
117. Haykin, S., *Neural Networks: A Comprehensive Foundation*. 1994: Prentice Hall PTR. 768.
118. Riley, G. and J.C. Giarratano, *Expert Systems: Principles and Programming*. 3 ed. 1998, Boston: PWS Publishing Company.
119. Graham, I., *Service Oriented Business Rules Management Systems*, I. Graham, Editor. 2005, TriReme: Pynton, Cheshire, UK. p. 52-75.
120. Hayes-Roth, F., *Rule-based systems.* Commun. ACM, 1985. **28**(9): p. 921-932.
121. Friedman-Hill, E., *Introducing Rule-Based Systems*, in *JESS in Action*. 2003, Manning Publications CO.: Greenwich. p. 1-28.
122. Feigenbaum, E.A., *Knowledge Engineering in the 1980's*, ed. D.o.C. Science. 1982, Stanford, CA.

123. Shang, Y., *Digital Systems and Computer Engineering*, in *The electrical engineering handbook*, W.-K. Chen, Editor. 2005, Elsevier Academic Press: Burlington. p. 367-377.
124. Abraham, A., *Rule-based Expert Systems*, in *Handbook of Measuring System Design*, P.H. Sydenham and R. Thorn, Editors. 2005, Wiley: Oklahoma, USA. p. 909-919.
125. Demey, J., M. Jarrar, and R. Meersman. *A Markup Language for ORM Business Rules*. in *International Workshop on Rule Markup Languages for Business Rules on the Semantic Web (RuleML-ISWC02 workshop)*. 2002. Sardinia, Italy: CEUR-WS Publication.
126. Mannan, S. and F.P. Lees, *Artificial Intelligence and Expert Systems*, in *Lees' Loss Prevention in the Process Industries: Hazard Identification, Assesment and Control*, S. Mannan and F.P. Lees, Editors. 2005, Elsevier Health Sciences: Burlington. p. 30.1-30.50.
127. Okafor, E. and C. Osuagwu, *Issues in structuring the knowledge-base of expert systems.* Electronic Journal of Knowledge Management, 2007. **5**(3): p. 313-322.
128. Newell, A. and H.A. Simon, *Human problem solving*. Vol. 104. 1972, Englewood Cliffs NJ: Prentice-Hall
129. Shortliffe, E.H., *Computer-based medical consultations, MYCIN*. 1976: Elsevier (New York) 264.
130. Christoph, N. *VIDRE--A Distributed Service-Oriented Business Rule Engine based on RuleML*. 2006.
131. Forgy, C.L., *On the efficient implementation of production systems*. 1979, Carnegie Mellon University. p. 187.
132. Forgy, C.L., *Rete: A fast algorithm for the many pattern/many object pattern match problem.* Artificial Intelligence, 1985. **19**(1): p. 17-37.
133. Brownston, L., et al., *Programming expert systems in OPS5*. 1985, Massachusetts: Addison-Wesley Pub. Company. Medium: X; Size: Pages: 471.
134. Rudolph, G. *Some Guidelines For Deciding Whether To Use A Rules Engine*. 2008; Available from: http://herzberg.ca.sandia.gov/jess/guidelines.shtml.
135. Mahmoud, Q.H. *Getting Started With the Java Rule Engine API (JSR 94): Toward Rule-Based Applications*. 2006; Available from: http://java.sun.com/developer/technicalArticles/J2SE/JavaRule.html.
136. Dubray, J.-J., *Business Process Metamodels and Services*, in *WorfFlow HANDBOOK 2005*, L. Fischer, Editor. 2005, Future Strategies: Lighthouse Point, Florida. p. 159-178.
137. Lawrence, C., *Integrated Function and Workflow*, in *WorfFlow HANDBOOK 2005*, L. Fischer, Editor. 2005, Future Strategies: Lighthouse Point, Florida. p. 31-43.
138. OMG. Available from: http://www.omg.org/.
139. WfMC. Available from: http://www.wfmc.org/.
140. Tagg, R., W. Lelatanavit, and S.S. Reddy. *Preliminary Design of a Lightweight Workflow Server*. in *Australasian Conference on Information Systems (ACIS)*. 1997. Adelaide, Australia.
141. Kenneth, R.A., *Experiences with workflow management: issues for the next generation.* CSCW '94: Proceedings of the 1994 ACM conference on Computer supported cooperative work, 1994: p. 113-120.
142. Wing, H., C. Liu, and R. Colomb, *A Bottom-up Approach to Distributed Workflow*, in *PACIS 1997*. 1997.
143. Becker, J., M. zur Muehlen, and M. Gille, *Workflow application architectures: classification and characteristics of workflow-based information systems*, in *Workflow handbook 2002*, L. Fischer, Editor. 2002, Future Strategies: Munster, Germany. p. 39-50.
144. Deelman, E., et al., *Pegasus: Mapping Scientific Workflows onto the Grid*, in *Grid Computing*, M.D. Dikaiakos, Editor. 2004, Springer Berlin / Heidelberg. p. 131-140.
145. Marin, M., *Business Process Technology: From EAI and Workflow to BPM*, in *WorfFlow HANDBOOK 2002*, L. Fischer, Editor. 2002, Future Strategies: Costa Mesa, USA. p. 133-145.
146. Merz, M., B. Liberman, and W. ERSDORF, *Using mobile agents to support interorganizational workflow management.* Applied Artificial Intelligence: An International Journal, 1997. **11**(6): p. 551 - 572.

147. Hollingsworth, D., *Workflow management coalition: The workflow reference model*. 1995, Workflow Management Coalition: Winchester Hampshire, UK.
148. WfMC, *Workflow Process Definition Interface – XML Process Definition Language*. 2002, Workflow Management Coalition: Hampshire, UK.
149. WfMC, *Terminology & Glossary*. 1999, Workflow Management Coalition: Hampshire, UK.
150. Hollingsworth, D., *An XML based Architecture for Collaborative Process Management*, in *WorfFlow HANDBOOK 2002*, L. Fischer, Editor. 2002, Future Strategies: Middlesex, United Kingdom. p. 95-103.
151. Savarimuthu, B.T.R., et al. *Integrating Web services with agent based workflow management system (WfMS)*. in *Web Intelligence, 2005. Proceedings. The 2005 IEEE/WIC/ACM International Conference on*. 2005.
152. Savarimuthu, B.T.R., et al., *Agent-based integration of Web Services with Workflow Management Systems*, in *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*. 2005, ACM: The Netherlands. p. 1345-1346.
153. WfMC, *The Workflow Reference Model*. 1995, Workflow Management Coalition: Hampshire, UK.
154. DeFee, J.M. and P. Harmon, *Business Activity Monitoring and Simulation*, in *WorfFlow HANDBOOK 2005*, L. Fischer, Editor. 2005, Future Strategies: Lighthouse Point, Florida. p. 53-58.
155. Nutt, G.J., *The evolution towards flexible workflow systems*. Distributed Systems Engineering, 1996. **3**(4): p. 276-194.
156. Herbst, J. and D. Karagiannis, *Integrating machine learning and workflow management to support acquisition and adaptation of workflow models.* Intelligent Systems in Accounting, Finance & Management, 2000. **9**(2): p. 67-92.
157. Khoshafian, D.S., *Narrowing the Semantic Gap between Business Process Analysis and Business Process Execution*, in *WorfFlow HANDBOOK 2005*, L. Fischer, Editor. 2005, Future Strategies: Lighthouse Point, Florida. p. 103-112.
158. Allen, R., *Workflow: An Introduction*, in *WorfFlow HANDBOOK 2001*, L. Fischer, Editor. 2001, Future Strategies: Brentford, Middlesex, UK. p. 15-38.
159. Prior, C., *Workflow and Process Management*, in *WorfFlow HANDBOOK 2003*, L. Fischer, Editor. 2003, Future Strategies: Sydney, Australia. p. 17-25.
160. WfMC, *Workflow Standard - Interoperability Abstract Specification*. 1999, Workflow Management Coalition: Winchester, UK.
161. WfMC, *Workflow Standard - Interoperability Wf-XML Binding*. 2001, Workflow Management Coalition: Florida, USA.
162. Mendling, J. and G. Neumann, *A Comparison of XML Interchange Formats for Business Process Modelling*, in *WorfFlow HANDBOOK 2005*, L. Fischer, Editor. 2005, Future Strategies: Lighthouse Point, Florida. p. 185-198.
163. Andrews, T., et al., *Business Process Execution Language for Web Services, Version 1.1*. 2003, IBM Corporation, Microsoft Corporation, BEA Systems, SAP AG, Siebel Systems.
164. Aberg, C., P. Lambrix, and N. Shahmehri. *An agent-based framework for integrating workflows and Web services*. in *Enabling Technologies: Infrastructure for Collaborative Enterprise, 2005. 14th IEEE International Workshops on*. 2005.
165. JADE. *Núcle de JADE Brasileiro*. 2008 [cited 2009; JADE brasil]. Available from: http://gos.tecnolink.com.br/.
166. FIPA. *FIPA Agent Management Specification SC00023*. 2002 [cited 2008; Available from: http://www.fipa.org/specs/fipa00023/SC00023K.html.
167. FIPA. *FIPA Agent Software Integration Specification XC00079*. 2002 [cited 2008; Available from: http://www.fipa.org/specs/fipa00079/XC00079B.html.
168. Hermenegildo, M.V. and K.J. Greene, *Prolog and its performance: exploiting independent and-parallelism*, in *Logic Programming*, D.H.D. Warren, Editor. 1990, MIT Press. p. 253-268.
169. XSB. *XSB - Logic Programming and Deductive Database System*. 2007 [cited 2010; Available from: http://xsb.sourceforge.net/index.html.

170. JBoss. *Drools Expert*. 2006 [cited 2009; Available from: http://jboss.org/drools/drools-expert.html.
171. Vrba, P., *JAVA-Based Agent Platform Evaluation*, in *Holonic and Multi-Agent Systems for Manufacturing*, V. Marík, D. McFarlane, and P. Valckenaers, Editors. 2004, Springer Berlin / Heidelberg. p. 1086-1087.
172. Bellifemine, F., et al., *JADE Programmer's Guide*. 2002.
173. Bellifemine, F., A. Poggi, and G. Rimassa, *Developing Multi-agent Systems with JADE*, in *Intelligent Agents VII Agent Theories Architectures and Languages*, C. Castelfranchi and Y. Lespérance, Editors. 2001, Springer Berlin / Heidelberg. p. 42-47.
174. Bellifemine, F., A. Poggi, and G. Rimassa, *JADE: a FIPA2000 compliant agent development environment*, in *Proceedings of the fifth international conference on Autonomous agents*. 2001, ACM: Montreal, Quebec, Canada. p. 216-217.
175. Dietrich, J., J. Hiller, and A. Kozlenkov. *The Mandarax Project*. 2007 [cited 2009; Available from: http://mandarax.sourceforge.net/.
176. Kozlenkov, A. and A. Paschke. *PROVA - Highly expressive distributed Semantic Web rule engine*. 2007 [cited 2009; Available from: http://www.prova.ws/.
177. Dietrich, J. and C. Crasborn. *TAKE - a Java rule compiler* 2007 [cited 2009; Available from: http://code.google.com/p/take/.
178. Filho, C.F. *JEOPS - The Java Embedded Object Production System*. 2007 [cited 2009; Available from: http://www.di.ufpe.br/~jeops/.
179. Beedle, M. *JLisa - A Clips-like Rule engine accessible from Java with the full power of Common Lisp*. 2003 [cited 2009; Available from: http://jlisa.sourceforge.net/.
180. Feldman, J. *Openules - Open Source Business Rules Management System*. 2003 [cited 2009; Available from: http://openrules.com/index.htm.
181. Doss, J. and P. Skangos. *OpenLexicon - open-source business rules and process management tool* 2007 [cited 2009; Available from: http://www.openlexicon.org/index.php.
182. Grosof, B., et al. *SweetRules: Tools for Semantic Web Rules and Ontologies*. . 2005; Available from: http://sweetrules.projects.semwebcentral.org/.
183. Levy, E. *Zilonis Rules Engine*. 2007 [cited 2009; Available from: http://www.zilonis.org/.
184. Hewett, M., J. Crawford, and B. Kuipers. *Algernon - Rule-Based Programming*. 2005 [cited 2009; Available from: http://algernon-j.sourceforge.net/.
185. Volder, K.D. *TyRuba (Type Rule Base): logic programming language* 2001 [cited 2009; Available from: http://tyruba.sourceforge.net/.
186. Frank, G. *JTP: An Object-Oriented Modular Reasoning System*. 2005; Available from: http://www.ksl.stanford.edu/software/JTP/.
187. JLog. *JLog - Prolog in Java* 2002; Available from: http://jlogic.sourceforge.net/.
188. Mindswap. *Pellet is an open-source Java based OWL DL reasoner*. 2004 [cited 2009; Available from: http://www.mindswap.org/2003/pellet/.
189. Nau, D. *SHOP - Simple Hierarchical Ordered Planner*. 2004; Available from: http://www.cs.umd.edu/projects/shop/index.html.
190. Group, H. *Hammurapi Rules*. Available from: http://www.hammurapi.com/dokuwiki/doku.php/products:hammurapi_rules:start

http://www.hammurapi.biz/hammurapi-biz/ef/xmenu/hammurapi-group/products/hammurapi-rules/index.html.
191. EsperTech. *Esper for Java*. 2006; Available from: http://esper.codehaus.org/about/esper/esper.html.
192. Koch, F. *mProlog: Lightweight PROLOG Engine*. Available from: http://www.cs.uu.nl/3apl-m/mprolog.html.
193. Shor, S. and S. Zyrianov. *OpenL Tablets*. 2006; Available from: http://openl-tablets.sourceforge.net/.
194. IBM, *ABLE Rule Language - User's Guide and Reference*. 2005, IBM.
195. IBM. *ABLE: Agent Building and Learning Environment*. 2000 [cited 2009; Available from: http://www.alphaworks.ibm.com/tech/able.

196. Brodt, R. *BPEL Designer Project*. 2004 [cited 2009; Available from: http://www.eclipse.org/bpel/index.php.
197. BPEL. *Oracle BPEL Process Manager*. Available from: http://www.oracle.com/appserver/bpel_home.html.
198. ODE, A. *Apache ODE: Orchestration Director Engine*. 2006; Available from: http://ode.apache.org/index.html.
199. Valdés-Faura, M. and B. Diard. *Bonitasoft: Bonita Open Solution - BPM and Workflow Software*. 2001; Available from: http://www.bonitasoft.com/.
200. Endpoints, A. *ActiveBPEL Engine*. 2003 [cited 2009; Available from: http://www.activevos.com/community-open-source.php.
201. Soika, R. *Imixs Workflow the open source workflow technology for business applications*. 2006 [cited 2009; Available from: http://www.imixs.org/.
202. Price, A. *OBE Open Business Engine*. 2002 [cited 2009; Available from: http://obe.sourceforge.net/.
203. JBoss. *jBPM - makes your work flow*. 2003 [cited 2009; Available from: http://www.jboss.org/jbpm.html.
204. Webb, S. *Sarasvati: A Sanskrit IME*. 2004 [cited 2009; Available from: http://code.google.com/p/sarasvati/.
205. Engels, H. and R. Malitz. *OSBL Con:cern*. 2003 [cited 2009; Available from: http://osbl.wilken.de/wiki/index.php/Con:cern.
206. Mettraux, o. *ruote - open source ruby workflow engine*. 2002 [cited 2009; Available from: http://ruote.rubyforge.org/.
207. Danet, D. *WfMOpen*. 2001; Available from: http://wfmopen.sourceforge.net/.
208. myGrid. *Taverna* 2006 [cited 2009; Available from: http://www.taverna.org.uk/.
209. YAWL. *YAWL: Yet Another Workflow Language*. 2004; Available from: http://www.yawlfoundation.org/.
210. RunaWFE. *Runa WFE*. 2004; Available from: http://wf.runa.ru/About.
211. Together. *Java XPDL Workflow Editor+Server*. 2009 [cited 2009; Available from: http://www.together.at/prod/workflow.
212. JBoss. *Drools Flow*. 2009; Available from: http://www.jboss.org/drools/drools-flow.